Journal of Computer Virology and Hacking Techniques manuscript No.
(will be inserted by the editor)

An Investigation of Byte N-Gram Features for Malware Classification

Edward Raff · Richard Zak · Russell Cox · Jared Sylvester · Paul Yacci · Rebecca Ward · Anna Tracy · Mark McLean · Charles Nicholas

Received: 30 March 2016 / Accepted: 6 August 2016

Abstract Malware classification using machine learning algorithms is a difficult task, in part due to the absence of strong natural features in raw executable binary files. Byte n-grams previously have been used as features, but little work has been done to explain their performance or to understand what concepts are actually being learned.

In contrast to other work using n-gram features, in this work we use orders of magnitude more data, and we perform feature selection during model building using Elastic-Net regularized Logistic Regression. We compute a regularization path and analyze novel *multi-byte identifiers*. Through this process, we discover significant previously unreported issues with byte n-gram features that cause their benefits and practicality to be overestimated. Three primary issues emerged from our work. First, we discovered a flaw in how previous corpora were created that leads to an over-estimation of classification accuracy. Second, we discovered that most of the information contained in n-grams stem from string features that could be obtained in simpler ways. Finally, we demonstrate that n-gram features promote overfitting, even with linear models and extreme regularization.

Keywords malware classification · byte n-grams · multi-byte identifier · elastic-net

The final publication is available at Springer via http://dx.doi.org/10.1007/s11416-016-0283-1

E. Raff · R. Zak · C. Nicholas
Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle Baltimore, MD 21250 USA
E-mail: raff.edward@umbc.edu
E-mail: richard.zak@umbc.edu
E-mail: nicholas@umbc.edu

R. Cox · J. Sylvester · P. Yacci · R. Ward · A. Tracey · M. McLean
Laboratory for Physical Sciences, 8050 Greenmead Drive College Park, MD 20740
E-mail: jared@lps.umd.edu
E-mail: mrmclea@lps.umd.edu

1 Introduction

Choosing appropriate and informative features is the prerequisite step to all effective applications of machine learning. Initial feature selection is a particularly difficult task for malware analysis due to the lack of natural or obvious features. The use of domain knowledge in feature extraction is often desirable, but complicated in this case by a malicious adversary that may intentionally corrupt or break standard rules.

Byte n-grams have been used as features in a number of works, and are one of the most common feature types used for static analysis [34]. By treating a file as a sequence of bytes, byte n-grams are extracted by looking at the unique combination of every n consecutive bytes as an individual feature. Most experiments from other works range from $n = 1$ to $n = 8$ bytes, and are generally reported to be effective for any $n \geq 2$ with papers determining various different values of n as performing the best [34]. Byte n-grams are particularly attractive since they require no knowledge of the file format, do not require any dynamic analysis, and could potentially learn information from both headers and the binary code sections of an executable[24]. The n-gram approach's lack of domain knowledge requirement means a byte n-gram system could be used for other file formats, such as PDFs, without being re-engineered.

Given these benefits combined with reported accuracies of 95% or better, we sought to investigate what

features are learned by byte n-grams and why they perform so well. In this paper we examine performance for $n \in \{4, 6\}$. We find 6-grams to perform best, and use them as the basis of our investigation into what concepts are actually being learned by our model.

1.1 Overview of our contributions

Some of the potential shortcomings with byte level n-grams in the realm of malware classification have been discussed before [35], but we are not aware of any work that attempts to assess their true effectiveness or generalization to new data sets. To do this assessment, we use multiple separate sources of data for our experiments, divided into two higher level "groups" and given an overview of this data in section 3.

We began our investigation by attempting to reproduce previous work, but our larger collection of data resulted in more potential features. For this reason we performed an evaluation of feature selection methods (section 4). As part of improving the feature-selection process we used Elastic-Net regularized Logistic Regression as our classifier, which performs implicit feature selection. In subsection 4.1, we examine both the final model performance, and the regularization path, where we discover significant over-fitting of our n-gram models and a possible methodological flaw in the data-collection process of some, if not most, previous papers.

In section 5 we investigate the nature of our features, and determine why they don't work as well as expected. We present evidence that our n-grams are learning string-like features rather than information from the code or other sections of an EXE file. Based on these results, we devise the Multi-Byte Identifier in subsection 5.1 as a technique to help further evaluate n-grams for EXE files. The final experiment in our investigation is covered in subsection 5.2, where we provide evidence that most of the *generalizable* information may be coming from ASCII strings. Given these surprising results, we discuss what we believe are the major weaknesses of the byte n-gramming approach in section 6.

2 Related Work

The problem of distinguishing malicious from benign executables has been long studied, with many earlier attempts relying on hand crafted features [19]. We have a particular interest in byte n-grams for this problem, as they require no domain knowledge to apply. This makes it feasible for a small team and code base to be used for a wide variety of file types, provided adequate training data can be obtained. Kephart et al. [16] provide one of the earliest instances of n-grams for malware analysis, using byte 3-grams to classify infected boot-sectors.

For the case of byte n-grams for Microsoft PE binaries, Schultz et al. [33] provide the earliest work of which we are aware. Shultz et al. considered DLL imports, Strings, and byte n-grams as features, evaluating them using a number of different classifiers. In their work a Naive Bayes classifier had the best overall accuracy at 97.1%, followed by an ensemble of Naive Bayes classifiers using byte n-grams at 96.9%. Their n-gram based model also had the highest detection rate of malware. Shultz et al. compared against a simple signature based approach, which achieved only 49.3% accuracy, showing the importance of expanding beyond signatures in defending against new, unseen malware. Abou-Assaleh et al. [1] made the connection with techniques in Natural Language Processing work and using a feature selection step, reporting 98% cross validation scores using a nearest neighbor classifier.

Kolter and Maloof [17, 18] looked exclusively at byte n-grams for classifying benign vs. malicious executables, as well as classifying malicious EXEs by payload method. They performed their initial work on a smaller set of 1,037 files, by which they settled on using 4-grams for their features and AdaBoost [8] with C4.5-style decision trees[28] consistently provided the best results. In addition they used Information Gain to prune the set of 4-grams down to the top 500 used for their model. This approach obtained an AUC of 0.984. Testing on a larger set of 3,622 files reached an impressive AUC of 0.996 for the benign vs malicious task. In attempts to explore the information captured by their model, Kolter and Maloof did discover evidence of string features being extracted by their model, but make no effort to quantify their significance to the overall model. We establish that strings make a significant portion of the information learned and discriminative power in section 5. Their work has been regularly replicated, for example, by Jain and Meena [15]. For comparison purposes, we replicate their approach in our work as well using 4 and 6-gram features.

Most work on using byte n-grams for malware classification follow the overall method set by Kolter and Maloof: choose a value of n, use some feature ranking scheme to select a few hundred (up to say one thousand) n-grams, and then evaluate (using cross validation or with a random training / testing split) with one or more classifiers. In our work we mostly eliminate the need for a feature selection step by choosing a model that performs feature selection implicitly, as discussed in section 4. Since their work, there has been a significant amount of follow up work using byte n-grams, often as

a major component in a larger system or improving a component of the process.

One line of research has been on improved feature selection, as we do by using the Elastic-Net regularized model discussed in subsection 4.1. Reddy and Pujari [30] looked at improving the ROC curve by performing a selection on benign and malicious n-grams independently, though no AUC is given. They note explicitly the difficulty in analyzing the selected n-grams, which we help to resolve in this paper. Henchiri and Japkowicz [13] sought to improve feature selection by weighting the feature selection to n-grams that occur in multiple virus families. Henchiri and Japkowicz also discuss some of the issues in using cross validation to estimate the true accuracy of a classification model for benign versus malicious EXEs, and adjust their cross validation so that each CV fold has all of a unique malware family in it. Our use of separate datasets tackles this same bias issue.

A number of works, such as Elovici et al. [7], Menahem et al. [23], and Masud et al. [21] have looked at combining n-grams with other features. Masud et al. fused byte n-grams, opcode n-grams, and DLL function imports into a larger classification system. They also evaluated on two datasets, similar to our approach discussed in subsection 6.1. Masud et al.'s two datasets have overlap with each-other, where ours are kept completely disjoint. In their work 4 and 6-grams performed almost equally and obtained 95.4% and 93.6% on their two datasets. The hybrid approach presented in their work obtained 96.3% and 97.6% for 6-grams respectively, indicating that while n-grams did not perform the best — they still performed well compared to an approach which required domain knowledge features. For feature selection and model construction Masud et al. used the same approach as Kolter and Maloof. We note as well that the assembly n-grams in Masud et al. perform worse than the byte n-grams for $n \geq 2$.

Another line of research, to which our work is a contribution, has included the use of larger datasets. Moskovitch et al. [25] pushed their dataset up to 30,000 files for n-gramming. Masud et al. [22] used a distributed system to process 105,388 files, obtaining an accuracy of 97.2% for their best model. This was achieved using 2000 4-grams selected by Information Gain, and using an ensemble of C4.5 decision trees.

Researchers have looked at building more sophisticated systems using byte n-grams as a significant component. Though not directly comparable, the information we discover about n-grams is relevant to the underpinning of these methods, since they are built upon byte n-grams. Perdisci et al. [27] developed a multi-stage system using both whole file byte n-grams and byte n-grams from binary code extracted via dynamic analysis, obtaining a final AUC of 0.977. Tahan et al. [37] looked at 3-grams to select and match "segments" which were used as their final features. Another related area that we do not compare directly against is the task of distinguishing between subfamilies of malware using byte n-grams, as is done by Zhang et al. [42] and Stolfo et al. [36].

3 Data

We take care to be explicit with how and where we obtained our data, as the data used had the most significant impact on performance. Our data is divided into two higher level groups, A and B, that are collected in different manners and from different sources. Every file in both datasets is a valid PE binary for either x86 or x64 Windows. We do not intermingle these data for training and keep them separate when testing so that we can better evaluate the generalization of our models. We do this because evaluating on held-out data that was collected in the same manner as training data may not adequately evaluate generalization, since both sets are biased by the same collection mechanism. Having separate sets collected in different ways helps avoid this issue. Since not all of our training and test sets have the same ratio of benign to malicious files, we always report a weighted accuracy such that the two classes have equal total weight in the score. Across all training and testing sets, no two samples in our data have the same MD5 checksum.

Group A's malware is taken from Virus Share [31], and we also use data from Open Malware [29] as a separate malware only test set. Group A's benign files (or "goodware") mostly come from various Microsoft Windows operating systems, including Windows XP, Window 7, and Windows 10. A smaller collection of goodware files were also downloaded from `portablefreeware.com` and from the Cygwin and MinGW installations. We use a held out test set from Windows 8.

We arbitrarily chose Windows 8 for the test set. We wanted to avoid having files from the same version of Windows in both the training and testing set to minimize any potential information leakage across the sets. We used as many files from Virus Share in the training set as we could given our computational resources, and used the remainder that we had downloaded for testing. The split of Virus Share in training and testing was random, since we had no additional information to improve the way data was split. During evaluation, Open Malware is reported as a separate line and is not included in the Group A numbers. Again, this is so we can better judge generalization since no data from Open Malware was in

the training set. The Open Malware data is collected in a manner similar to the Virus Share data, so we would expect their data to be more similar to each-other than our other data.

The use of a public malware corpus combined with Windows files and a selection of other commonly installed applications is the same strategy used to build the training and testing sets in most previous work [e.g., 13, 17, 25, 27, 33]. Our results for models trained on the Group A data should be representative of the results other works would have obtained.

During initial testing, models trained on Group A were not performing well on new data, despite encouraging cross validation scores. We sought out an industry partner to share data of a higher quality and more representative of the larger population, and is the source of our Group B data. Santos et al. [32] also used a private corpus from an industry partner, sampling 1,000 benign and malicious files. We received data from our partner in two batches. The testing set represents the first batch of data obtained, while the second and larger batch was used as the training set.

Both groups of data were randomly sampled from a larger set of benign and malicious EXEs that are meant to be representative of what is often seen on desktop computers (excluding Microsoft EXEs). The total number of files for all our data, Groups A and B, can be found in Table 1.

Table 1 Breakdown of the number of malicious and benign training and testing examples in each data group, along with the sources they were collected from. "Misc." comprises `portablefreeware.com`, Cygwin and MinGW.

	training		testing	
Group A	malicious	benign	malicious	benign
Virus Share	175,875	—	43,967	—
Open Malware	—	—	81,733	—
MS Windows	—	268,236	—	21,854
Misc.	—	1,195	—	—
total	*175,875*	*269,431*	*125,700*	*21,854*
Group B				
Industry Partner	200,000	200,000	40,000	37,349
total	*200,000*	*200,000*	*40,000*	*37,349*

4 Feature Selection and Model Building

An issue not adequately addressed in previous work is the feature selection process. For the 400k total files in Group B's training set, there are 4,289,759,510 unique 4-grams and 35,953,973,975 unique 6-grams observed. Storing the 6-grams naively, with 32 or 64-bit integers for count information, would take 503 or 791 GB of RAM, respectively. This issue alone is a significant road-block to applying byte n-gram features in practice.

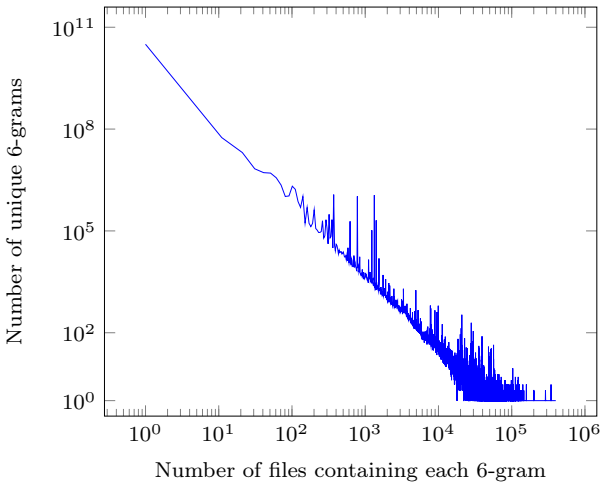

Fig. 1 From training-set of Group B, the number of 6-grams that occurred in x many files.

The feature selection process is made somewhat easier by the frequency of individual n-grams, as shown in Figure 1. We observe that they tend to follow a power-law type distribution, with 87.72% of 6-grams occurring only once, 97.58% 6-grams occurring ten or fewer times, and 99.61% with 100 or fewer occurrences. This is not surprising, since n-grams from NLP applications tend to follow a power-law (Zipfian) distribution as well. We felt the presence of such a distribution was worth confirming, since there is no reason n-grams would have to follow such a distribution when applied to different domains. We can reduce our set of candidate n-grams in general by selecting a minimum number of occurrences based on coverage in our dataset. For example, selecting 6-grams that occur in at least 1% of the aforementioned 400k files results in just under 1.6 million 6-grams (a reduction of more than 99.99%).

Learning from 1.6 million 6-grams is still a computational burden and provides strong potential for overfitting, so additional feature selection is necessary. Most previous work [e.g., 15, 27] use Information Gain criteria (1) or some other simple ranking scheme to select a fixed subset of n-grams. Our approach is to first do a coarse feature selection down to 200k n-grams, followed by a final feature selection during model construction.

We compared a number of ranking schemes to choose a subset of 200k n-grams which we list and briefly describe. For the equations below, g_j indicates the presence of n-gram j, and m_j and b_j are the number of malware and benign files that had g_j present. M denotes the mal-

ware class, B denotes the goodware or "benign" class, and $\mathrm{P}(x)$ is the probability of x given the training data. N_M and N_B indicate the number of n-grams found in malicious and benign files respectively. We tested each of the below methods, such as Information Gain (1), to select the initial 200k subset of n-grams.

$$IG(g_j) = \sum_{v \in \{g_j, \neg g_j\}} \sum_{C \in \{\mathrm{M,B}\}} \mathrm{P}(v, C) \cdot \log_2 \left(\frac{\mathrm{P}(v, C)}{\mathrm{P}(v) \cdot \mathrm{P}(C)} \right) \tag{1}$$

As an alternative to Information Gain, we introduced two simple scores that prefer features occurring in only one of the classes. Malice Score (2), which is biased toward features more common in malware and Benign Score (3) to favor features found in goodware.

$$\text{Malice Score}(g_j) = \mathrm{P}(g_j|\mathrm{M}) - \mathrm{P}(g_j|\mathrm{B}) \tag{2}$$

$$\text{Benign Score}(g_j) = \mathrm{P}(g_j|\mathrm{B}) - \mathrm{P}(g_j|\mathrm{M}) \tag{3}$$

To test favoring lop-sided occurrence rate in either direction we added the Absolute Malice Score (4). A simple variant on the Absolute Malice Score is Root Malice Score (5), which prefers more "pure" features based on the class in which it occurred.

$$\text{Absolute Malice Score}(g_j) = |\, \mathrm{P}(g_j|\mathrm{M}) - \mathrm{P}(g_j|\mathrm{B}) \,| \tag{4}$$

$$\text{Root Malice Score}(g_j) = \left| \sqrt{\mathrm{P}(g_j|\mathrm{B})} - \sqrt{\mathrm{P}(g_j|\mathrm{M})} \right| \tag{5}$$

We also evaluated ranking based on the Gini coefficient (6) and KL-divergence (7). Our Gini tests add c artificial observations to each n-gram, because without this modification many millions of n-grams all received the same maximal score.

$$\mathrm{Gini}_c(g_j) = \frac{2(m_j + c)(b_j + c)}{(m_j + b_j + 2c)^2} \tag{6}$$

$$\begin{aligned} \mathrm{KL}(g_j) = & \frac{m_j}{m_j + b_j} \cdot \log_2 \frac{m_j(N_M + N_B)}{N_M(m_j + b_j)} \\ & + \frac{b_j}{m_j + b_j} \cdot \log_2 \frac{b_j(N_M + N_B)}{N_M(m_j + b_j)} \end{aligned} \tag{7}$$

Much previous work used feature ranking schemes like these to do all of their feature selection. A shortcoming of this approach is the need to then estimate how many features to select. Prior works usually selected only a few hundred to one thousand n-grams, and then trained a model on the selected subset. Determining the appropriate value of k becomes its own expensive process, as noted in Kolter and Maloof [17] where the number of n-grams was chosen based on analysis of a subset of the data. In our approach (detailed in 4.1), the initial coarse feature selection is mostly for computational convenience. This is because we have chosen a Machine Learning model that does implicit feature selection as part of the model building process.

To compare these feature selection methods, we built models using each of them according to the method described in subsection 4.1 and sorted the models by their cross validated (CV) accuracy. The results are shown in Table 2. We see that simply selecting the n-grams that occurred in the most files performed best, with most methods resulting in only a minor difference. Specifically, all of the tested feature selection methods, except the Gini measure (6) and KL-divergence (7), obtained 90%+ accuracy. We note that of the simple sorting methods we tested — equations 2 through 5 — the Root Malice Score did slightly worse than the others. The square root term in (5) causes a slight preference for purity of label, that is to say the equation prefers to select n-grams that occur only in benign or malicious files but not both. This is a property shared by (6) and (7). Overall this would seem to indicate a need for common, high-frequency n-grams in order for our models to perform well.

Much of the previous work [17, 25] in using n-grams has suggested the use of Information Gain (1) based on its success in text classification and other NLP domains. Our results indicate that while Information Gain does work well, we can use much simpler approaches by choosing a model that has feature selection built-in to the model's training.

Table 2 10-fold CV accuracy rate on group B training data using the top 200k features selected by different methods. Test set errors had similar ranking.

Selection Method	CV Accuracy
Frequency	96.6%
Malice Score	96.3%
Abs Malice Score	96.3%
Benign Score	96.3%
Info Gain	95.2%
Root Malice Score	94.6%
Gini_{32}	85.0%
KL	78.7%
Gini_{256}	77.2%

A downside not addressed in previous works is that adding more data does not significantly change the number of n-grams that become viable model features, causing diminishing returns as data is added. This is counter to the argument presented in Schultz et al. [33]. For example, for the results of our whole model building process: a model built from the training set of Group B obtains a weighted accuracy of 87% on all data, excluding Open Malware, from Group A. Training from the much smaller test-set of Group B gets an accuracy of 84.4% on the same data. Evaluating the recall on only the Open Malware data, the performance only drops from 81% to 80%. This puts n-grams in a poor place, as doubling the amount of data can double the amount of resources required but produce only marginal improvement in outcome. This makes it difficult to exploit the phenomena that adding more data tends to provide significant improvements to accuracy [3, 6, 12], which is what we would normally expect. The minor increase in accuracy could be an indication that our features do work well or are nearing the representational capacity of our features and model. We don't believe this is the cause, as we observe evidence of overfitting in the remainder of this section, as exemplified in Table 3 and discussed through the rest of this paper.

4.1 Elastic-Net Models of N-Grams

We build our models with Logistic Regression using either of two different regularization methods: Lasso (also called L_1) [38] and Elastic-Net [43]. The objective function of both can be defined using equation 8: for Lasso, $\alpha = 1$ and for Elastic-Net, $\alpha = 0.5$. The value C in the loss function is our regularization parameter. Larger values of C decreases the strength of the regularization. As $C \to \infty$, (8) approaches the behavior of standard Logistic Regression. Smaller values of C reduce the flexibility and effective degrees of freedom of the model, encouraging the solution vector w to approach zero.

$$f(w) = \alpha||w||_1 + (1-\alpha)||w||_2 + C\sum_{i=1}^{N}\log(1+\exp(-y\cdot w^{\mathsf{T}}x_i)) \tag{8}$$

We use these methods for two reasons. First, they provide a principled method of feature selection that is robust to extremely high dimensional data with many irrelevant features [26] while also being computationally tractable. Their feature selection property is a direct result of the model building process, as exact zeros will occur in the optimal solution of w as a result of the $\alpha||w||_1$ term in the objective function. Second, we can compute a regularization path, where we look at the properties of the model (e.g. accuracy, number of non-zero weights, or coefficient values) as a function of C. This regularization path provides insights into the model's performance and the quality of the selected features.

For our training process, we start with the 200,000 6-grams based on total number of files they occurred in (i.e., an n-gram is counted only once per file). Feature vectors are constructed as binary vectors, with zero indicating the absence of the feature in the file, and one indicating its presence.

We also experimented with using a larger number of features to start (up to one million), with using TF-IDF style weightings for occurrences, and with counting n-grams by total number of occurrences in a file when considering the top selection. None of these experiments resulted in a noticeable change in accuracy, or the behavior of the regularization path when trained. Any one of these changes could result in a different subset of n-grams being selected by our final model, with up to 50% of them changing. We are concerned that up to half of the selected features could change with no discernible impact on accuracy and believe this is an indication of the weakness of byte n-gram features.

To create our regularization path, we need a minimum and maximum value for C to consider. We first compute C_0, the value of C that would result in a weight vector of all zeros as specified in Yuan et al. [40]. We then use a starting value of $2\,C_0$ as the strongest regularization we evaluate, and set the weakest regularization to be $2\,C_0 \times 10^5$. We compute the regularization path along 300 logarithmically spaced points along this range, following the basic warm-start strategy in Friedman et al. [9]. The warm-starting strategy allows us to build these models sequentially at only incremental cost. However, our search is over 5 orders of magnitude, where Friedman's search only covered 2 orders of magnitude. We use this larger range due to unusual behavior observed in the regularization path, and highlight the issues below. We confirmed our results using two implementations of Elastic-Net Logistic Regression, one an extension of the new GLMNET [41] to the Elastic-Net case and another using the extension to OWL-QN presented in Gong and Ye [10].

In Figures 2 and 3, we show the number of non-zero values in the weight vector and the accuracy over the regularization path using 6-gram features. These results give us several reasons to suspect that our n-gram features are over-fitting the data.

Our data requires extreme levels of regularization to obtain an informative regularization path, which is the main reason we use such a large search range for C. In Yuan et al. [41] the smallest regularization value

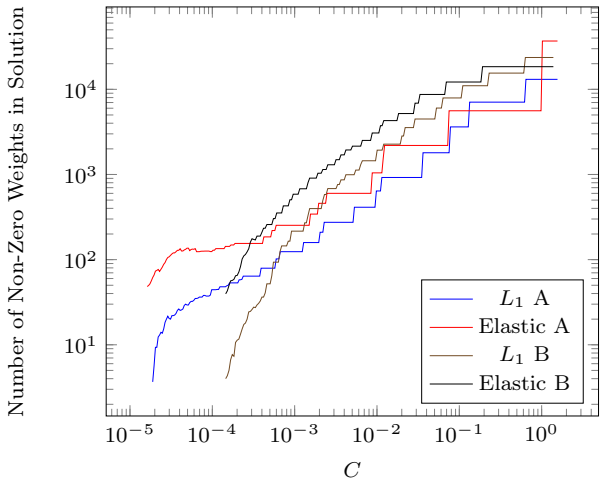

Fig. 2 Average number of non-zero weights in solution vector based on 3-fold cross validation across regularization path.

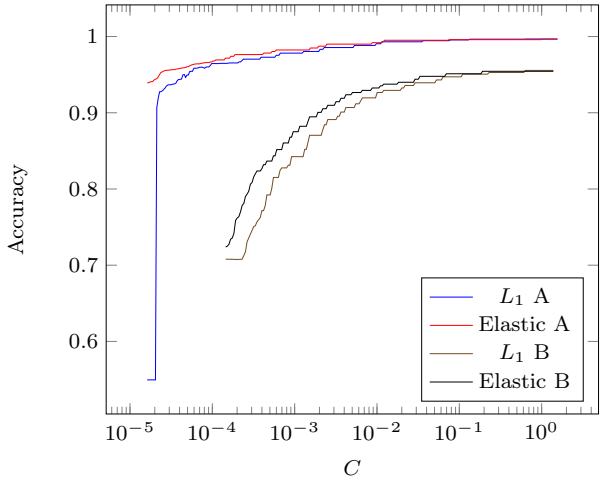

Fig. 3 Average accuracy based on 3-fold cross validation across regularization path.

considered is 2^{-4}, which is already near the maximum of our evaluated range and at the point of diminishing returns. If we had used the ranges suggested in other papers our regression model would have exhibited almost no change in the weight vector for all tested values.[1]

Even if we ignore the abnormally high amount of regularization, the behavior of the regularization paths of both Group A and Group B models are irregular. Looking at Figure 2, the most pronounced issue is the step-ladder addition of features in large groups at a time, rather than more continuous additions of features a few at a time. This behavior is obvious in the Group A models, but also occurs in the Group B models. In Figure 3, the Group A models exhibit higher accuracies over the whole range of regularization, much higher than we would expect. The differences between the models' CV error rate gives a strong indicator of data quality issues.

In addition, we note that the L_1 Group B model in Figure 3 has a CV accuracy of 71% using an average of only four features. The L_1 Group A model gets 91% using just ten. The Elastic-Net Groups B & A models use a similarly small 40 features to obtain 72% and 94% accuracy respectively. A priori, it seems highly unreasonable that as few as ten 6-grams should be able to obtain such high accuracies. Both Group B models are entering a plateau of 95% accuracy by 10,000 features, and both Group A models plateau of 99%+ by just 2,000 features. If n-grams were effectively learning features from the binary sections of an executable, it seems unlikely that millions of malware and goodware EXE files would be forced to use such a small subset of binary code. We confirm in section 5 that the n-grams are not effectively learning binary features.

Table 3 Performance of Kolter and Maloof (KM), L_1 and Elastic-Net regularized models trained on both groups of data, and applied to all testing data. Open Malware (OM) is recall, others are weighted accuracy. Column indicates which data was used for training the model used. Row indicates the test set and whether 4 or 6-gram features were used.

		Group A			Group B		
		L_1	Elastic	KM	L_1	Elastic	KM
6-gram	OM	96.9%	97.2%	96.2%	81.1%	81.2%	49.5%
	A Test	99.7%	99.6%	99.1%	87.3%	87.0%	72.8%
	B Test	68.1%	68.1%	62.0%	94.5%	92.5%	85.5%
4-gram	OM	97.0%	97.3%	97.3%	67.1%	64.1%	52.6%
	A Test	99.6%	99.6%	99.3%	84.9%	84.7%	74.3%
	B Test	70.5%	68.8%	65.8%	90.6%	89.7%	86.6%

The final testing accuracy is shown in Table 3. We use a weighted accuracy so that the malware and benign samples in the test set have equal total weight. Since the Open Malware test set comprises only malicious files, we list the recall rate rather than accuracy. The Open Malware files are not included in the A Test scores.

In our results the models trained on Group A do not generalize to the data in Group B, getting an accuracy much lower than what was achieved in all previous works. Due to the bias in how Group A's data was collected, we believe that the Group A model has actually learned an "is it a windows executable?" classifier, rather than "is it malware?". This would explain the high recall on Open Malware. The model learns to say "no, not Windows" (i.e. malware) for all the data, since none of it came from Windows and the whole collection is malware. However, saying "no" for the Group B goodware (which also did not come from Windows) results in an error rate approaching random guessing. The lopsided errors

[1] In extended testing, neither the accuracy or number of non-zeros increased when testing larger values of C.

in the Group B test set corroborate that the models defaults to classifying most inputs as malware by default, and then use features present to switch to a decision of goodware. This is further confirmed by looking at the false-positives on the portablefreeware, Cygwin, and MinGW files. Since these were a part of the training set, we would expect them to be predicted correctly by the model. However we found that MinGW and Cygwin had a false positive rate of 39%, and portablefreeware data had a false-positive rate of 43%. These values are extraordinarily high and do not reflect the Group A test set accuracies, providing strong evidence that the model is not accurately learning the desired concept.

Models trained on Group B generalize better to Group A testing than vice versa, though the accuracy on the A test set is lower than the Group B test set, and the recall on Open Malware is down to the low 80s. Despite the drop, this overall behavior is consistently better — as it is generalizing past the training distribution. This indicates the 6-gram model trained on this data has meaningfully captured some information. The spread in test set performance, combined with the behavior exhibited in Figures 2 and 3 give us caution that there may still be some level of over-fitting occurring. This is corroborated by the higher CV accuracies in Table 2, and suggests that the I.I.D. assumption of cross validation is too strongly violated in our corpora to be used for evaluation. Given the similarity of our Group A data to others, and its increased scale, this brings many previous results into question — especially when used as the only validation as in Abou-Assaleh et al. [1]. We also note that the Group B test performance drops to around 81% accuracy when considering the generalization to the Open Malware data. While this does not necessarily represent the accuracy we would expect when deploying this model to various users, it is considerably below the mid to high 90s that are reported by most others. This indicates that the true effectiveness of binary n-grams for malware classification has been considerably over-estimated.

Comparing to the baseline approached used in Kolter and Maloof [17] (KM), our use of Elastic-Net and Lasso regularized Logistic Regression has provided a dramatic performance improvement when trained on the Group B data. Because KM use boosted decision trees, which can learn non-linear decision surfaces, we know the difference in accuracy is not a capacity issue of the chosen model since we used a simpler linear model. The KM approach trained and tested on Group B performs worse, likely due to a lack of features. By using a model with the feature selection process built in, our performance has improved generalization and simplified the feature selection task. We also note that the KM's approach loses generalization accuracy at a quicker rate when moving from the B test data to A test, and from A test to Open Malware. Considering the extreme levels of regularization we are using for a linear model, it is likely that the KM approach's difficulty in generalizing past the training data is caused by the use of a more powerful and flexible discriminative model. That is to say, the non-linear model used in KM has greater capacity to overfit the data — which we believe to be the issue since our simpler model is also showing evidence of overfitting.

4-gram vs 6-gram Performance

On the issue of n-gram size, we note that there is no significant difference in the overall behavior of the regularization path between 4 and 6-grams, though there is an apparent difference in generalization accuracy. In Figure 4 (corresponding to Figure 2) we see the same general pattern of the Group B models initially selecting more features than the Group A models, before reaching a common plateau. In Figure 5 (corresponding to Figure 3) we again see that the models trained on Group A immediately reach a higher accuracy, with the Group B models not reaching as high and taking longer to reach their plateau. We note that the cross validation scores on the Group B data are reaching higher accuracies for 4-grams (97.4%) than was obtained for the 6-grams (95.6%). Yet 4-grams have lower test set accuracies than 6-grams in Table 3. This means the 4-grams are exhibiting even higher degrees of overfitting compared to 6-grams.

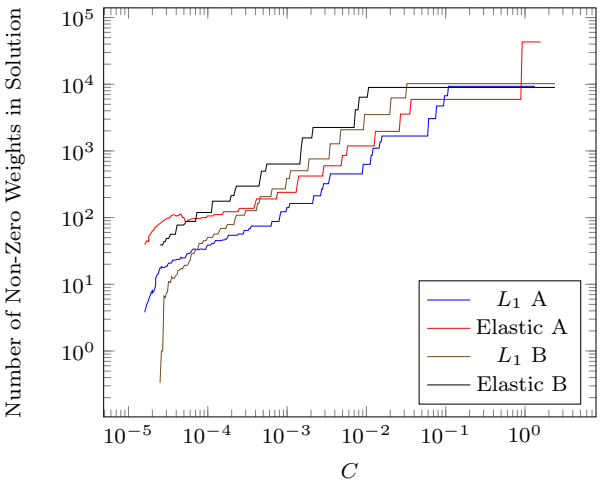

Fig. 4 Average number of non-zero weights in solution vector based on 4-gram features and 3-fold cross validation across regularization path.

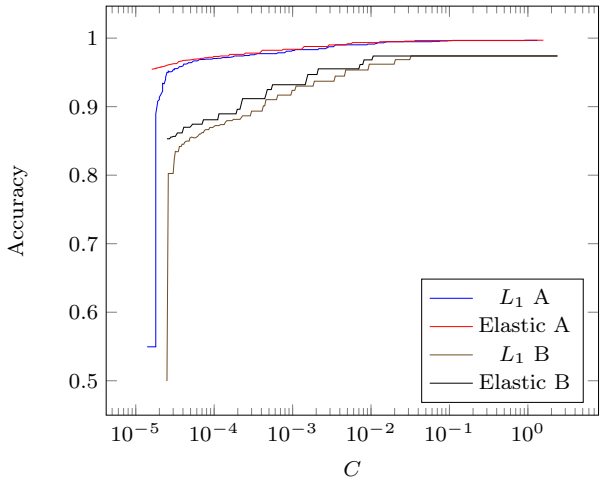

Fig. 5 Average accuracy based on 4-gram features and 3-fold cross validation across regularization path.

Looking at the numbers in Table 3, we see that despite having the same general behavior — 6-grams trained on Group B generalized considerably better than 4-grams to the Open Malware data, with 4-grams trailing by 14 percentage points. The difference is not as large for the A and B test sets, but the 6-grams do continue to perform better by 2 to 4 points. This would seem to validate our general preference for 6-grams over 4-grams. Looking only at the models trained on Group A, the difference in accuracy mostly disappears. Considering that the Group A models are overfitting to the Microsoft Windows data, it is understandable that their performances would converge. This would also explain why most prior works, using data similar to Group A, have shown little performance difference for n-grams when testing various values of n.

5 N-Gram Evaluation

Given the poor level of performance compared to previous results when given more data, we sought to evaluate what the n-grams extracted were actually learning. The starting point of this was to use the 6-grams selected by the Elastic-Net model trained on Group B to obtain some simple statistics. We use the 6-gram model since 6-grams had better performance than 4-grams in subsection 4.1.

First, we look at the entropy of where our 6-grams occurred. Because different modalities of information have different average entropies [20], looking at these statistics gives us an idea of what information may be captured by the n-grams. Since 4 and 6-grams are too small to compute a meaningfully entropy measure, we estimate the entropy of an n-gram from a window of bytes around where it occurred in a file. We compute the Shannon entropy (9), where p_i is the proportion of byte i in the given window.

$$S = -\sum_{i=1}^{256} p_i \cdot \log(p_i) \tag{9}$$

In our testing we used a window of 128 bytes, though results were not sensitive to exact window size. The distribution of n-gram entropies are presented in Table 4. 26% of our n-grams occurred in entropy regions more often associated with plain-text (i.e. $S \leq 4$) [20]. From this table, it would appear that up to 74% of our n-grams are occurring in entropy regions associated with executable code (that may be packed or encrypted). This is initially encouraging, as it indicates our 6-grams do occur in regions associated with code and hence, may be learning to discriminate between benign and malicious code.

We also looked at how well distributed our features are among the test data. In Figure 6 we plot the fraction of 6-grams the model selected that occurred in the Group B testing data. If our 6-gram features are working effectively, we would like to see a relatively even distribution of feature occurrences, but the unfortunate trend is that our 6-grams are not evenly distributed through the test data. Instead a few files have a significant fraction of features present, quickly trailing off toward almost none of the features being present. For n-grams to generalize well, we need to see our features occur in new files on a consistent and regular basis, as they are intrinsically critical to making the classification decision. The likelihood of observing our n-grams can only decrease when tested on data from a different source, making this trend a significant issue.

Table 4 Percentage of n-grams that occurred within an entropy window (rounded to nearest integer).

Entropy	Proportion	Cumulative
0	0.4%	0.4%
1	1.5%	1.9%
2	3.8%	5.7%
3	11.1%	16.8%
4	9.2%	26.0%
5	19.2%	45.2%
6	48.3%	93.5%
7	5.7%	99.2%
8	0.8%	100%

These statistics give us some higher level information, but do not explain any of our results. Since so few features are needed to obtain high CV accuracies, and based on the proportion of our n-grams occurring in low entropy regions, we examined the ASCII decoding of the

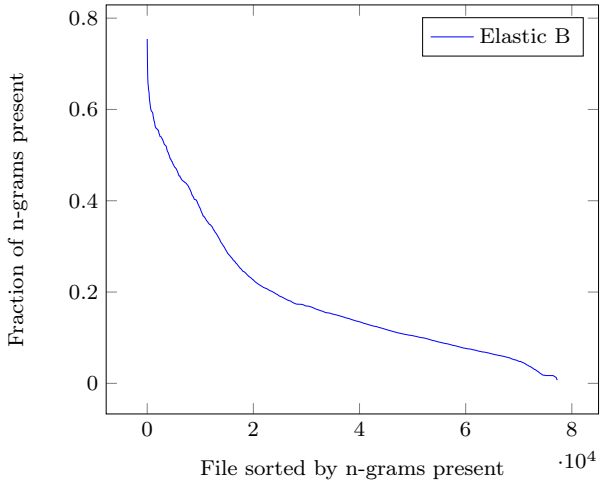

Fig. 6 Fraction of 6-grams used in the model that are observed in each testing point.

n-grams selected at the beginning of the regularization path. A subset of the 4 and 6-grams selected by the L_1 regularized models are presented in Table 5. Sixteen 6-grams and 4-grams were selected by the models trained on Group A; thirteen 6-grams and 27 4-grams were selected for the models trained on Group B. Note, the number of non-zero features selected is larger than what is shown in Figure 2 because those numbers are the average number of non-zeros from the CV models, whereas this is the actual model trained on all data at that regularization strength.

Looking at the ASCII decodings, we can see that the models trained on Group A are selecting parts of the text "Microsoft Corporation", which is embedded in most of the executables that come with any installation of Microsoft Windows, often if not always with symbols such as "®" and "©". Because L_1 regularization does not like selecting highly correlated features [43], it has difficulty obtaining the n-grams to complete the string.

The model trained on Group B 6-grams appears to be picking up items from the header and import tables, selecting most of the import "KERNEL32.DLL" and a `GetProcess` function. By looking at the Elastic-Net 6-grams, since it has no issue selecting correlated features, we can confirm that our hypothesis is correct and that this behavior extends out into the beginning of the regularization path's search (such tables can be found in the appendix). The model trained on Group B 4-grams is still picking up string information, but seems to prefer some different information. It does not tend to select strings that make function imports, like the 6-grams do. We suspect this is an issue with the smaller 4-grams matching too many other tokens as well, losing some of their discriminative ability. The items selected by the 4-grams are generally selected by the 6-gram model later in the regularization path. Overall, the most discriminative items being selected appear to be string features. This result appears to contradict Schultz et al. [33], which posited that n-gram features provide additional robustness to the model since they are harder to avoid than string-based features.

Table 5 Selection of 4 and 6-grams chosen by the most strongly regularized L_1 models. Whitespace denoted using the '␣' symbol

Group A		Group B	
6-gram	ASCII	6-gram	ASCII
20004D006900	␣Mi	000047657450	GetP
00720070006F	rpo	657450726F63	etProc
006F00720070	orp	00004C6F6164	Load
43006F007200	Cor	6B65726E656C	kernel
000100560061	Va	00004B45524E	KERN
004400000001	D	4C33322E444C	L32.DL
4-gram	ASCII	4-gram	ASCII
4D006900	Mi	69726541	ireA
7400AE00	t®	6C3D2272	l="r
00720070	rp	3C736563	<sec
20004300	␣C	2E637274	.crt
00010056	V	696F6E3E	ion>
72794100	ryA	3C2F7365	</se

5.1 Multi-Byte Identifiers

Based on the behavior of the n-grams observed in Table 5 we develop the hypothesis that informative malicious or benign segments in the data are longer than our n-grams. If this is the case, we should see sequences of adjacent or overlapping n-grams when processing our files, and it would explain why increasing n to even higher values does not tend to cause any significant drop in performance. We call any such sequence a Multi-Byte Identifier, or MBI. We search for MBIs using the subset of n-grams chosen by our Elastic-Net model, since any larger sequences of n-grams will be highly correlated and thus less likely to be selected by Lasso.

It is difficult to manually evaluate how n-grams are being used as more are added to the model. MBIs can help us to better understand what higher level concepts are being learned by our model. After manually inspecting MBIs from a few files, we used a simple heuristic to separate MBIs into three categories: Strings, Simple, and Other. Examining the contents and statistics of these categories helps us better understand what types of information are being captured by our n-gram models.

1. **Strings**: Any MBI where more than half of the bytes are ASCII printable characters.

2. **Simple**: Any MBI whose hex representation has two or less distinct ASCII characters.
3. **Other**: Any MBI not in the other two categories.

Using this simple strategy, we find that 16.8% of all extracted MBIs are strings, 43.0% are simple, and 40.2% belong to "other". If restricted to unique MBIs, we get 20.6%, 26.8%, and 52.7% respectively. The changes in percentage of MBIs come mostly from the simple section, which has the most repetition. Of the string MBIs, most are learning to find items like "GetProcAddress" and "WIN32.DLL" that are detecting imports or section names such as "`.text`" and "`.rsrc`".[2] Some strings appeared in data or code sections, an example being "PADDINGXX" repeated 26 times. Most simple MBIs were interleaved sequences of *0x00* and *0xFF* of varying lengths, sometimes over 1 KB in size. We were not able to ascertain the nature of the other MBIs. An attempt was made to disassemble these MBIs, but they did not always produce meaningful assembly. While we were able to occasionally find useful MBIs in this category,[3] we were not able to determine how meaningful this larger category was as a whole.

These results indicate that n-grams may not be learning strong features. The percentage of MBIs that are very simple is especially concerning given how frequently they appear in both benign and malicious files. We believe many such MBIs, like *0xFF00FFFF000000*, are an artifact of the overfitting we have observed. In generalizing to new data, they would then act as noise in the decision process. We note that there is the possibility for meaningful MBIs in the simple category, such as *0x0C0C0C0C* [5] or the sequence *0xCCCCCCCC* for repeated `int3` calls to interfere with a debugger. Unfortunately we did not tend to see these in the MBIs our models learned, though they do exist in our data set.

Objectively, any MBIs found in the code section of a file will be brittle, especially if they call any function, as the addresses called can easily change based on changes in the header. In this case, our n-grams would act more as a signature for previously seen malware rather than a feature to predict novel malware. The accuracy on Group B's test set could be partially explained by the model learning a set of smaller signatures, which are applicable to test data from the same collection. While this could be useful for systems such as the one in Griffin et al. [11], signatures are intrinsically not generalizable features. Given the number of string MBIs and the n-grams chosen by our models in Table 5, we suspect that n-grams are obtaining most of their generalizable information from the PE header and plain-text strings present in the file, similar to the features used in Shafiq et al. [35]. We also note that while $n = 6$ bytes is large enough to capture the instruction and operands for about 97.6% of instructions[14], a valid x86 instruction can be up to 15 bytes in length. The variable-length nature of the instructions appears to be a general mismatch for fixed-length n-grams.

We were able to obtain the same MBI results using the n-grams from our L_1 regularized models as well. We hypothesize that this is an artifact of the lack of informative features. The L_1 model, as C increases, is forced to select increasingly larger groups of correlated features simultaneously. This would explain the unusual regularization paths in Figure 2 and the plateau in Figure 3 and that we are able to obtain MBIs with Lasso. However more testing is needed to be conclusive. Given these results, it may be interesting to explore using MBIs as features themselves in future work.

5.2 String Features

To determine how much of the discriminative power comes from n-grams picking up on string features, we perform another test using the occurrence of strings as features. We use the GNU `strings` command to extract all ASCII strings ≥ 4 characters from our training sets. We then create a regularization path using Lasso and Elastic net in the same way we did for n-grams, and plot the cross-validated accuracy as well as the test accuracies on Groups A and B as a function of C. For comparison, in each plot are two dashed lines representing the performance of the 6-gram features on the A and B test sets.

Once trained, we note that both models used a comparable number of string features as compared to the n-gram features. Both models trained on Group A strings selected around 5,500 out of 200,000 strings and the models trained on Group B selected around 24,000. In Figure 7 we see the cross-validated and test set accuracies of training an L_1 regularized model on group A training data using string features. As C increases, we see the test set accuracies reach the same values as the n-gram results. However, in the same plot in Figure 8 we see the CV score go above previous results, while the test set scores both fail to reach the same accuracies achieved when trained with n-grams.

Based on these results, it seems reasonable to conclude that n-gramming does learn some non-string features, but can degrade to learning only string features when given poor data. This further calls into question

[2] Amusingly, we discovered one instance of malware using the non-standard "`.virus`" section name

[3] An interesting example is the MBI *0x33C9B104014C24*, which we discovered is used by ClamAV as a signature (see `https://github.com/eqmcc/clamav_decode/blob/master/db/daily.ndb#L363`).

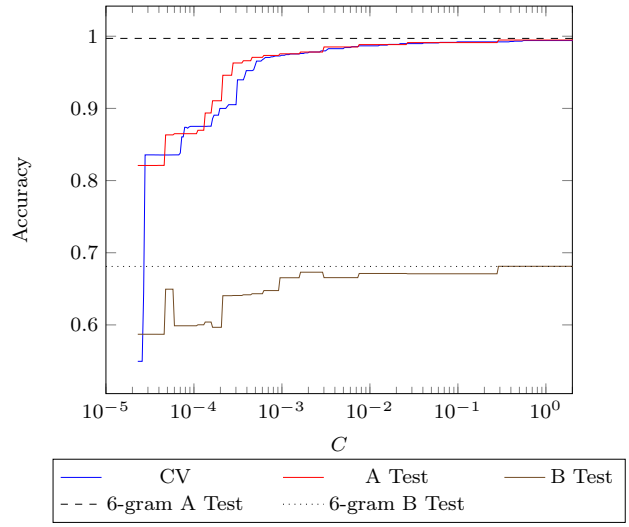

Fig. 7 3-fold CV and test set accuracy using string features. Trained using L_1 regularization from group A data

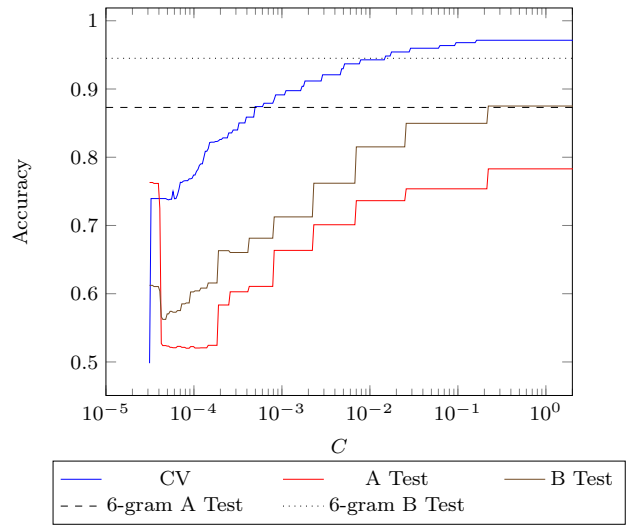

Fig. 8 3-fold CV and test set accuracy using string features. Trained using L_1 regularization from group B data

all previous results collected in the same manner as our Group A data. The gap in performance between n-grams and strings features trained from Group B does open the possibility that n-grams are learning some useful features that are not strings.

Another potential explanation of the performance gap is that exact extraction of strings fails to capture certain information that is captured by n-grams of strings. Some sub-strings may be more generalizble then the whole string they come from. Many of our strings are also sub-strings of larger strings we extracted. A most-common-substring may better capture their relationship and match other instances appended with characters that we had not seen, and therefore would not catch. Normalizing by common sub-string could also move items up in total count and therefore warrant consideration in the model, when it would have been removed for being too infrequent before. A similar issue we see is strings that are mostly the same, but have an edit distance of one or two. In some sense, n-grams learning from strings better handle these problems by limiting themselves to a fixed size, though we must rely on the model building process to adequately select enough components of these strings and properly weight them. We have not yet tested this hypothesis as a detailed analysis of string-based features is beyond the scope of this work.

6 Discussion

We have performed a significant investigation into the performance of n-grams, and found them to be learning less and performing worse than previous work would have suggested. We hypothesize that n-grams, at least used on the whole executable, have a number of intrinsic issues that have not been adequately discussed.

First is the discovered issue that n-grams appear to be learning mostly from string content in an executable, and items from the PE header (which also contains strings). We believe this is an issue intrinsic to n-grams. While there are billions of potential n-grams, we need to select the features that occur frequently enough to occur in new data as well. Selecting with a predisposition to frequency then encourages us to select lower entropy features, which consist primarily of strings and padding. However padding alone is not a particularly strong signal, and as mentioned, strings could have been obtained in a much simpler fashion.

As mentioned in subsection 5.1, a possible preference for 6-grams may be that they are large enough to regularly capture a whole x86 instruction, but still fall short 2.4% of the time. This makes processing with n-grams problematic in creating a mismatch between our features and the data. This is important since, assuming no instruction in isolation is an indicator of maliciousness, we need to capture multiple instructions in a single feature. To even attempt to consistently capture three sequential assembly instructions we would need to produce and process n-grams of at least 12 bytes in length for most cases, and up to 45 for extreme cases. This alone is simply too computationally demanding a task to perform. Even if we had the computational resources to do so, there would still be a problem of trading off between specificity and generalization. We want our features to be large enough that they are not likely to occur by chance in a new file, but small enough that they are not specific to the training data as this would degenerate into a signature-based approach, which would have

maximum specificity but no generalization. When considering the aforementioned frequency bias, it becomes even harder to imagine a large n-gram being selected by the model.

Another intrinsic issue is the loss of location information when using n-grams. Given our observed performance and the common use of string features, we believe that this limitation is likely playing a role in their weakness. For example, in our Group A malware we observed that at least 5% of the data had inconsistent section names compared to the permissions set in the PE header (e.g. a section named `.data` but marked as executable in the header).[4] The occurrence of a feature in a non-executable section, when normally it would be found in an executable section, could have a significantly different meaning that would never be recognized by n-gram approaches. This type of location mismatch could occur both based on a section's string name in the header, and what the section's true identity is based on the permissions set.

Finally, we believe that the drop in generalization performance from our Group B model can be explained in part by an intrinsic brittleness to n-gram features. Regardless of what our n-grams learn, to apply them we must obtain an exact match when classifying a new datum. This means any minor change will make the feature "disappear" in terms of its impact on our model. Consider that we see `GetProcAddress` as a common MBI in Group B, indicating that this Windows function is a common feature of malware. We would then like to see our n-grams learn to identify the `call` or `jmp` instructions, followed by the address of the `GetProcAddress` function as a feature. However, in the import table to the PE the address of any function can be arbitrary defined, thus making it impossible for any of our n-grams containing a `call` or `jmp` plus an address to generalize to new files.

The other part to the lack of generalization is an intrinsic potential for over-fitting that will be found with n-grams. Estimating model parameters when the number of features is near or larger than the number of samples is a classic scenario for over-fitting due to the curse of dimensionality [2, 4, 39]. Since each file is one sample and based on the power-law observation of the n-gram distribution, we should reasonably expect that every new executable will give us thousands to tens of thousands of previously unobserved n-grams (depending on the size of the file). Thus we are in a scenario that will always produce many more features than samples. We selected Lasso and Elastic-Net for their robustness in this scenario and many of the features can be removed by frequency counts, but these techniques are not entirely immune to the curse of dimensionality.

[4] We did not perform an exhaustive test for consistency; this was merely a property found when testing some simple hypotheses. The true number could be much higher.

6.1 On Using Multiple Training and Testing Sets

Because the issue of overfitting is critical to our results we also discuss why we have used multiple, independent datasets for our evaluation, which may seem unusual at first. We do this because the space of possible malware and goodware is extraordinarily large, and it is not possible for us draw a well-distributed random sample of real files from this space. Executables from Microsoft Windows are the most readily available source of benign files. Since these all come from the same source, they all share a strong common bias. Another collection strategy is to include commonly installed applications, such as third-party web browsers, as a source of goodware. This obtains only a few thousand EXEs, which is not enough to represent the larger population of different miscellaneous applications people may have. (If such limited data were an adequate representation of the class of goodware then it would be easy enough to construct a white list of known safe applications.)

It is easier to obtain large amounts of Malware thanks to resources such as Virus Share and Open Malware, and from the use of honeypots. Yet these sources are also biased. They are run by volunteers, and the samples are provided by volunteers. So each source already has an intrinsic bias in the malware that is provided by those who make the effort to do so. There may be individuals or organizations that see malware and are not able or willing to share it, and thus that data won't make it into the collection. Private data collections used by anti-virus corporations, such as our industry partner, are also biased by their data collection mechanism and contractual agreements with clients and customers.

Given all of these potential biases and the large space of possible EXEs, we must be careful in our inference of the true generalization of our results. Using multiple evaluation sets that avoid sharing common biases helps us to better determine generalization. For example, if Group A was sufficient to learn the benign versus malware problem, we would expect a model's cross validated accuracy on Group A and test set accuracy on Group B to be similar; however, it is not. Said another way, we want to see models trained on one dataset generalize to new datasets. If the model is able to generalize to a completely new dataset, that gives us higher confidence it will continue to perform well on new and novel malware. If it fails to do so, then we have little confidence that the model will continue to perform well against new and novel malware.

If we instead merged all our data into one larger dataset, and performed cross validation, the aforementioned biases will be in all folds, and we would obtain little information about the true generalization of the model. This is an issue with the assumption that data is independently and identically distributed. While this assumptions is rarely entirely true in practice, it is violated in too strong a manner for cross validation to be valid for our data. If it were appropriate to merge our Group A and B data and use cross validation, we would expect to see similar performance across datasets and similar features learned. We have shown in subsection 4.1 that the performance is not consistent across datasets, and in section 5 that the features learned are different depending on the training dataset.

7 Conclusion

Previous work has reported byte n-grams to be highly effective for malware classification. The goal of our work was to investigate this feature type and determine how it performed so well on such a difficult task. By applying and evaluating Elastic-Net and L_1 regularized Logistic Regression and the novel Multi-Byte Identifier to the n-gramming of a corpus of previously un-reported size, we have shown significant issues with the use of n-gram features. They are computationally expensive, exhibit diminishing returns with more data, are prone to over-fitting, and do not seem to carry information much stronger than what is more readily available from the PE header and ASCII strings. While n-grams do have some merit as a feature for executable files, their results have been significantly over-estimated in the literature. Overall, it seems that we could obtain the same performance using much simpler and more interpretable techniques. We have also observed a larger issue, to wit, many papers are evaluating with a corpus of benign files collected mostly from Windows installations, which is too simple a subset to accurately represent the goodware vs malware problem. Our work highlights the importance of investigating what a model has actually learned, rather than simply accepting held out or cross validated scores as true performance.

The information present in n-grams are still not fully explored, and may still be useful in a more restricted context. For example, while using only n-grams is not very effective, can n-gramming a limited portion of an executable be informative in conjunction with other features? It is also an open question as to whether n-gramming with domain knowledge could improve results, where n-grams extracted from the `.text` section of a EXE file are processed and treated differently then n-grams from other sections. Of course this may nullify one of the prime advantages of n-gramming, to wit, that it requires no domain knowledge to apply. Future avenues of research include evaluating the viability of n-grams in other domains, and exploring what other approaches may work for EXE files but require limited or no domain knowledge. Based on the information we have extracted, we plan to do an investigation in the use of n-grammed disassembled instructions combined with the header information that were used by the n-grams. The results based on our multiple data sources also leads to a larger open question: how do we evaluate the quality of an executable corpus?

References

[1] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proc. 28th annual int'l computer software & applications conference*. Vol. 2. IEEE, 2004, pp. 41–42.

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Spaces. In *Proc. 8th int'l conf. on database theory*. J. v. d. Bussche and V. Vianu, editors. Springer-Verlag, 2001, pp. 420–434.

[3] M. Banko and E. Brill. Scaling to Very Very Large Corpora for Natural Language Disambiguation. In *Proceedings of the 39th annual meeting on association for computational linguistics*, 2001, pp. 26–33.

[4] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[5] Corelan Team. Exploit writing tutorial, part 11: heap spraying demystified. 2011. URL: `https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/` (visited on 05/25/2016).

[6] P. Domingos. A Few Useful Things to Know About Machine Learning. *Commun. acm*, 55(10):78–87, Oct. 2012. ISSN: 0001-0782.

[7] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer. Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic. In *Proceedings of the 30th annual german conference on advances in artificial intelligence*. In KI '07. Springer-Verlag, Berlin, Heidelberg, 2007, pp. 44–50. ISBN: 978-3-540-74564-8.

[8] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning (ICML 1996)*. L. Saitta, editor. Morgan Kaufmann, 1996, pp. 148–156.

[9] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1–22, 2010.

[10] P. Gong and J. Ye. A modified orthant-wise limited memory quasi-Newton method with convergence analysis. In *Proc. 32nd int'l conf. on machine learning*. Vol. 37, 2015, pp. 276–284.

[11] K. Griffin, S. Schneider, X. Hu, and T.-C. Chiueh. Automatic generation of string signatures for malware detection. In R. Lippmann and A. Clark, editors, *Recent advances in intrusion detection*, pp. 101–120, 2009.

[12] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *Intelligent systems, ieee*, 24(2):8–12, 2009.

[13] O. Henchiri and N. Japkowicz. A Feature Selection and Evaluation Scheme for Computer Virus Detection. In *Proc. of the 6th int'l. conf. on data mining*. IEEE Computer Society, 2006, pp. 891–895. ISBN: 0-7695-2701-9.

[14] A. H. Ibrahim, M. B. Abdelhalim, H. Hussein, and A. Fahmy. Analysis of x86 instruction set usage for Windows 7 applications. In *2nd int'l conf. on computer technology & development*, Nov. 2010, pp. 511–516.

[15] S. Jain and Y. K. Meena. Byte level n-gram analysis for malware detection. In K. R. Venugopal and L. M. Patnaik, editors, *Computer networks and intelligent computing*, pp. 51–59. Springer, 2011.

[16] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White. Biologically Inspired Defenses Against Computer Viruses. In *Proceedings of the 14th international joint conference on artificial intelligence*. Vol. 1. Morgan Kaufmann, 1995, pp. 985–996. ISBN: 1-55860-363-8.

[17] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of machine learning research*, 7:2721–2744, Dec. 2006.

[18] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proc. of the 2004 ACM SIGKDD int'l conf. on knowledge discovery and data mining*. ACM Press, 2004, pp. 470–478.

[19] R. W. Lo, K. N. Levitt, and R. A. Olsson. Refereed Paper: MCF: A Malicious Code Filter. *Comput. secur.*, 14(6):541–566, Jan. 1995. ISSN: 0167-4048.

[20] R. Lyda and J. Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE security and privacy magazine*, 5(2):40–45, Mar. 2007.

[21] M. M. Masud, L. Khan, and B. Thuraisingham. A scalable multi-level feature extraction technique to detect malicious executables. *Information systems frontiers*, 10(1):33–45, Mar. 2008.

[22] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Cloud-based malware detection for evolving data streams. *ACM transactions on management information systems*, 2(3):1–27, Oct. 2011.

[23] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici. Improving Malware Detection by Applying Multi-inducer Ensemble. *Comput. stat. data anal.*, 53(4):1483–1494, Feb. 2009. ISSN: 0167-9473.

[24] Microsoft Portable Executable and Common Object File Format Specification Version 8.3. Tech. rep. Microsoft, 2013, p. 98.

[25] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, N. Japkowicz, and Y. Elovici. Unknown malcode detection and the imbalance problem. *Journal in computer virology*, 5(4):295–308, Nov. 2009.

[26] A. Y. Ng. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Proc. 21st int'l conf. on machine learning*, 2004, pp. 78–86.

[27] R. Perdisci, A. Lanzi, and W. Lee. McBoost: boosting scalability in malware collection and analysis using statistical classification of executables. In *Annual computer security applications conference (ACSAC)*. IEEE, Dec. 2008, pp. 301–310.

[28] J. R. Quinlan. *C4.5: programs for machine learning*. Vol. 1(3) of *Morgan Kaufmann series in Machine Learning*. Morgan Kaufmann, 1993. ISBN: 1558602380.

[29] D. Quist. Open malware. URL: `http://openmalware.org/` (visited on 05/25/2016).

[30] D. K. S. Reddy and A. K. Pujari. N-gram analysis for computer virus detection. *Journal in computer virology*, 2(3):231–239, Nov. 2006.

[31] J.-M. Roberts. Virus share. URL: `https://virusshare.com/` (visited on 05/25/2016).

[32] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas. N-grams-based file signatures for malware detection. In *Proc. 11th int'l conf. on enterprise information systems*, 2009, pp. 317–320.

[33] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In *Proc. IEEE symposium on security and privacy*, 2001, pp. 38–49.

[34] 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*, 14(1):16–29, 2009. ISSN: 1363-4127.

[35] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. PE-Miner: mining structural information to detect malicious executables in realtime. In R. Lippmann and A. Clark, editors, *Recent advances in intrusion detection*, pp. 121–141, 2009.

[36] S. J. Stolfo, K. Wang, and W.-J. Li. *Towards stealthy malware detection*. In. *Malware detection*. M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, editors. Springer, 2007, pp. 231–249. ISBN: 978-0-387-44599-1.

[37] G. Tahan, L. Rokach, and Y. Shahar. Mal-ID: automatic malware detection using common segment analysis and meta-features. *Journal of machine learning research*, 13:949–979, Apr. 2012. ISSN: 1532-4435.

[38] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the royal statistical society B*, 58(1):267–288, 1994.

[39] M. Verleysen and D. François. The Curse of Dimensionality in Data Mining and Time Series Prediction. In *Proc. 8th int'l conf. on artificial neural networks: computational intelligence and bioinspired systems*. J. Cabestany, A. Prieto, and F. Sandoval, editors, 2005, pp. 758–770.

[40] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale L_1-regularized linear classification. *Journal of machine learning research*, 11:3183–3234, 2010.

[41] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for L_1-regularized logistic regression. *Journal of machine learning research*, 13:1999–2030, 2012.

[42] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang. Malicious codes detection based on ensemble learning. In *Proceedings of the 4th international conference on autonomic and trusted computing*. Springer-Verlag, 2007, pp. 468–477. ISBN: 3-540-73546-1.

[43] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society B*, 67(2):301–320, Apr. 2005.

Table 6 Percentage of Group B data, and Group A Malware, marked as malware by ClamAV. Cases where ClamAV and the label disagreed were uploaded to Virus Total to help confirm the label quality.

	Group B Train		Group B Test		Group A Train	Group A Test
Label	Goodware	Malware	Goodware	Malware	Malware	Malawre
ClamAV says Malware	0.4%	81.2%	1.4%	78.3%	66.6%	66.5%
Virus Total 0 AVs say Malware	82%	—	12%	—	—	—
Virus Total [1,5] AVs say Malware	12%	—	22%	—	—	6%
Virus Total [6,15] AVs say Malware	6%	—	36%	12%	2%	—
Virus Total [16,25] AVs say Malware	—	96%	24%	10%	2%	—
Virus Total 26+ AVs say Malware	—	4%	6%	78%	96%	94%

Appendix A Group B Data Labels

Since the Group B data is not publicly available, we attempt to provide extra details about the contents for readers who are interested. The results of the n-gram analysis convince us that the resulting model has more utility than one constructed from Group A style data, as many prior works have done. One may wonder then how much of the generalization gap incurred by the Group B model is due to differences in the data distribution, rather than the weaknesses of byte n-grams, or potential label errors. Fully answering such a question is beyond the scope of this work, however we hope the additional details may be of use to the reader to better understand the results as a whole and dissuade any concern of label quality.

To obtain a rough estimate of label errors we ran all Group B data (goodware and malware) through ClamAV.[5] We chose ClamAV because it is freely available to everyone and usable on all operating systems. If the labels are correct, we expect to see that most goodware is not flagged by ClamAV and that most malware is. Because ClamAV is not the most advanced of anti-virus (AV) products, we upload a random sample of 50 files in which ClamAV's output disagreed with our labels to Virus Total[6] for further confirmation. Virus Total uses an ensemble of anti-virus systems to asses each sample it is given. This produces more reliable labels, as well as giving us a proxy for confidence in those labels: a file which is identified as malicious by 20 of Virus Total's constituent programs is more likely to be malignant than one identified by only a single anti-virus program. While it would have been preferable to upload all of our data to Virus Total, a rate limit on the API means it would take multiple months to process the entire corpus. The results of this examination can be seen in Table 6. The first line shows the percentage of files marked as malware by ClamAV, and the remaining lines are the statistics of potentially mislabeled data sent to Virus Total.

We can see for the goodware Group B data, ClamAV marks almost all files as benign, with the test dataset having a higher conflict of 1.4% of the data. Even if all of such data was in fact mislabeled malware, the percentage would be small enough that a robust machine learning system should be able to learn from it. From the sample sent to Virus Total, we can see that most files had no or only a handful of anti-virus systems flag the files, giving us confidence that ClamAV was throwing false positives on the goodware data it identified as malicious. We note as well that anti-virus products may, in general, throw false positives for more challenging benign samples. A benign application with more sophisticated code (e.g, for performing encryption, just-in-time compilation, or disk formatting) may seem malicious to an AV product for good reason, despite having no malicious purpose as an application. We have observed that a number of the Group B test goodware files, marked by multiple AVs, are products for encryption that do not have a clear malicious intent. We avoid explicitly naming these products due to privacy concerns.

While the Group B goodware samples sent to Virus Total are more anomalous compared to those from the training set, we are still confident in the majority of the labels since we obtained. If we assume that all samples marked by 6 or more AVs is in fact malware, then Group B's test set goodware would contain only 0.9% mislabeled files. The maximal change in test accuracy is thus less than one percentage point, and then would not meaningfully impact any of the results or conclusions of our work.

For the malware datasets, ClamAV fails to recognize a much higher percentage as malicious: around 19% of files for Group B and 33% for Group A. When uploading samples that were not caught by ClamAV, every sample was marked by at least one anti-virus. Most were marked by a plethora of products (26 or more), with only a handful being marked by less than 10. This again gives us

[5] `https://www.clamav.net/`

[6] `https://www.virustotal.com/`

confidence that our labels are correct, and that ClamAV is throwing false negatives.

As mentioned in our conclusion, the construction and evaluation of a high quality dataset for this task is still an open problem. We believe we have provided ample evidence that our Group B data is of a better quality than the datasets normally used (i.e., Group A type data), but there is room for improvement in validating and constructing yet larger corpora for this task.

Table 7 ClamAV labels found in only the malware portion of the Group A and Group B data. "Assorted Other" includes obscure labels that had 5 or fewer occurrences in any dataset.

ClamAV Label	B Train	B Test	A Train	A Test
Dos.Trojan	0	0	18	4
BC.Win.Trojan	0	4	32	8
BC.Win.Virus	217	26	0	0
Heuristics.Encrypted	0	1	9	4
Heuristics.Trojan	1	10	400	104
Heuristics.W32	1	478	59	15
Js.Adware	16	0	0	0
Html.Trojan	0	14	0	99
Legacy.Tool	0	0	9	2
Legacy.Trojan	140	133	604	145
Pdf.Exploit	0	1	8	2
Win.Adware	77,613	7,551	1,924	454
Win.Downloader	306	353	14,250	3,599
Win.Dropper	74	67	2,834	672
Win.Exploit	15	16	588	180
Win.Ircbot	0	0	83	15
Win.Joke	0	0	63	26
Win.Keylogger	0	0	7	4
Win.Malware	10	3	5	4
Win.Proxy	0	0	148	31
Win.Spyware	75	285	8,579	2,089
Win.Tool	12	1	455	110
Win.Trojan	83,701	18,672	80,246	20,033
Win.Virus	52	573	177	42
Win.Worm	191	3,132	6,181	1,589
Assorted Other	5	1	15	6

We also look at the labels ClamAV produces for the files it does recognize as malignant in the malware data. These are shown in Table 7. We caution that the labels should not be taken as an absolute ground truth; ClamAV did not recognize a significant percentage of each dataset as malware, and anti-virus products in general do not always agree on the type of, or label for, an individual datum. It is also important to note that some of the labels are quite unexpected, indicating JavaScript, PDF, and HTML malware. We reiterate that all files in all of our datasets are valid PE files. These labels are either faulty in their designation, or an indication of our executables containing malware of a considerably different nature would be anticipated.

Looking at the Group A and B data separately, and comparing within group train and test sets, we see fairly consistent patterns. Looking across groups, we do see some distributional similarities and differences. Both Groups A and B are comprised mostly of Trojans, and do contain a significant amount of Adware. In Group B, Adware is a close second for the malware type, where Droppers make a far second for Group A. Group A seems to have, in general, a wider array of malware types. It is possible that the differences in distribution account for some portion of the decrease in generalization when applying a model trained on Group B to data from Group A. At the same time, it is also important for a model to generalize to novel data, which in this case includes malware of a type never seen before.

Appendix B String Features

We include a few examples of the string features found. We place these in the appendix as they have not yet been fully analyzed or understood for their meaning. Some were very long, and despite occurring many times in our training data are thus unlikely to occur in new novel malware. For example, the string "*Cl8lgpHNMm FBrO1rzHOloWOdjXN98PMI9inkJeYt8SxmqihhSpfFj 7VOP5e{BYbBZHRNwyPknt93{P3mPuCYDO6G{ew Blbc9gLOQ*" occurred 175,725 times in 780 files from the Group B training malware. A number of strings were simply the full function or DLL import names that n-grams were trying to extract, such as *KERNEL32.dll*, *Sleep*, *ExitProcess*, and *MultiByteToWideChar*. We also see many strings at least claiming that the malware is appropriately signed, like *1(c) 2006 VeriSign, Inc. - For authorized use only1E0C*. There are a number of variations of different strings like this, where it may have been better to select *VeriSign* as our feature rather than the larger strings. N-grams implicitly do select this smaller component rather than the text that happens to be around it.

There are also simpler cases of the sub-string issue. One example is the URL *http://crl.comodoca.com/COMODOCodeSigningCA2.crl0r*, which is found with several different one-character differences appended to this base URL. Another important note is that we see many full URLs in the string data, and a better solution would detect and process URLs differently. There are also instances where the differences are a simple pattern of replacement, such as *protocol_not_supported* versus *protocol not supported*. There are also many patterns where processing and removing repetition would collapse many complicated strings into one or a few base strings. For example a sequence of commas like "„„" occurs many times with differing amounts of repetition.

B.1 Extra String Graphs

The plots using string features (Figures 7 and 8) were relatively similar when using Elastic-Net or L_1 regularization. Figures 9 and 10 show the same plots for Elastic-Net for comparison. Also included is the average number of non-zeros added to the solution as C increased (Figure 11).

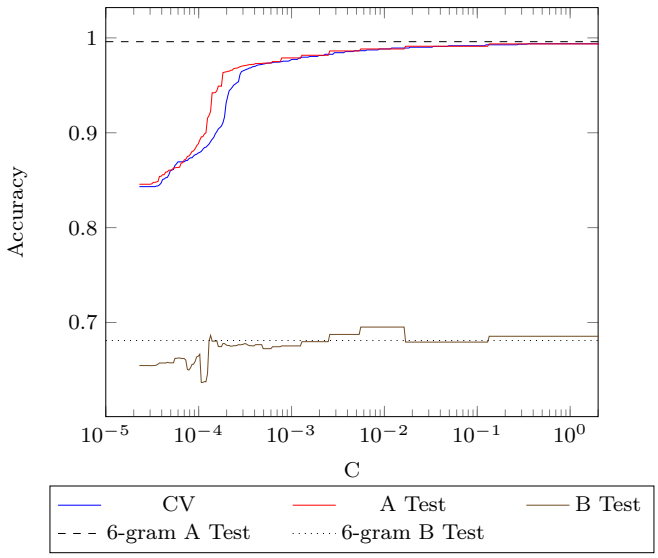

Fig. 9 3-fold CV and test set accuracy using string features. Trained using Elastic-Net regularization from group A data.

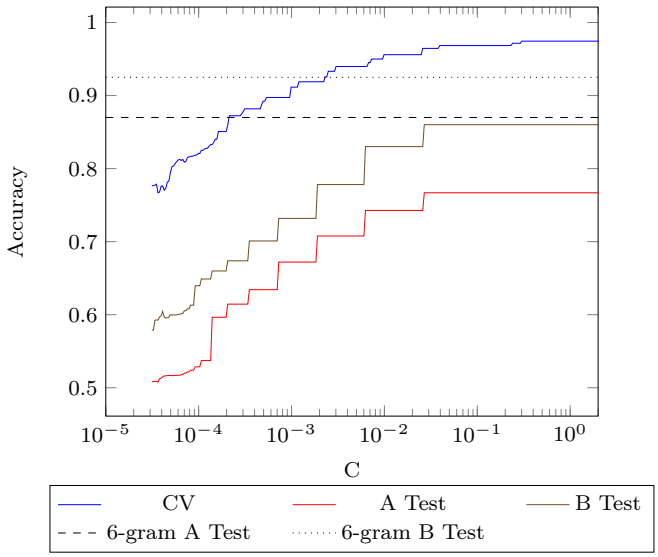

Fig. 10 3-fold CV and test set accuracy using string features. Trained using Elastic-Net regularization from group B data.

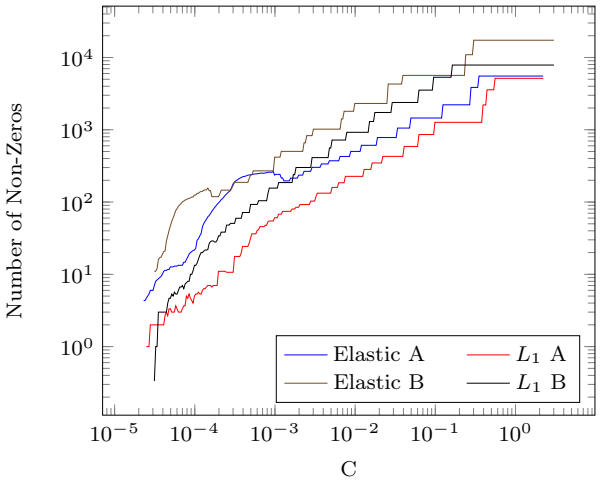

Fig. 11 Average number of non-zero weights in solution vector based on 3-fold cross validation across regularization path. Feature space is string occurrences.

Appendix C N-Gram ASCII Decodings

Below we list some of the n-grams selected by models early in the regularization path. The features selected earliest in the path are those that the model found most predictive of the 200k starting set. For legibility, we remove some n-grams from some of the larger tables. The n-grams removed either had no ASCII printable tokens, or had the same ASCII printout when decoded as another n-gram. In this case, the n-grams generally differed in being shifted over one byte by the value 0x00. We depict whitespace characters using the '␣' symbol. Only two tables for 4-gram models are included since they are more difficult to read and interpret.

Table 8 13 6-grams selected by Lasso on group B

6-gram	ASCII decoding
0000200000E0	␣?
000000E0000F	?
726300000000	rc
4C33322E444C	L32.DL
400000C02E72	@?.r
00004B45524E	KERN
6B65726E656C	kernel
00004C6F6164	Load
657450726F63	etProc
000047657450	GetP
000040000010	@
400000100000	@
004000001000	@

Table 9 16 6-grams selected by Lasso on group A

6-gram	ASCII decoding
4C0016000100	L
0000A9002000	?␣
74000000A900	t?
004400000001	D
000000010056	V
000001005600	V
010056006100	Va
000100560061	Va
006F00720061	ora
43006F007200	Cor
0043006F0072	Cor
6F0072007000	orp
006F00720070	orp
720070006F00	rpo
00720070006F	rpo
20004D006900	␣Mi

Table 10 26 6-grams selected by Lasso on group A

6-gram	ASCII decoding
6F61644C6962	oadLib
322E646C6C00	2.dll
726172794100	raryA
000040000010	@
400000100000	@
004000001000	@
690070007400	ipt
66007400AE00	ft?
000000004D00	M
000000A90020	?␣
0074000000A9	t?
0000A9002000	?␣
74000000A900	t?
004400000001	D
000000010056	V
000001005600	V
010056006100	Va
000100560061	Va
006F00720061	ora
43006F007200	Cor
0043006F0072	Cor
6F0072007000	orp
006F00720070	orp
720070006F00	rpo
00720070006F	rpo
20004D006900	␣Mi

Table 11 Subset of 58 6-grams selected by Elastic-Net on group B. Some non-printable and repeated ASCII 6-grams removed for legibility

6-gram	ASCII decoding
46696C654E61	FileNa
745665727369	tVersi
200028007800	␣(x
757465784100	utexA
000400040001	
4500004C0103	EL
00004C010300	L
30000000E404	0?
813950450000	?9PE
000002003030	00
C38B65E88B75	Ëe?u
FF0000FFFFFC	????
000058000080	X?
5F5E8BC35BC9	_^??[?
000300010000	
000088000080	??
000000004B45	KE
020000050001	
0000200000E0	␣?
0000E0000F01	?
00E0000F010B	?\n
007800650000	xe
000000E0000F	?
59C38B65E88B	YËe?
656C3D227265	el="re
76656C3D2272	vel="r
6C3D22726571	l="req
000001000300	
717569726541	quireA
726541646D69	reAdmi
756972654164	uireAd
69726541646D	ireAdm
726300000000	rc
454C33322E44	EL32.D
4B45524E454C	KERNEL
45524E454C33	ERNEL3
524E454C3332	RNEL32
4C33322E444C	L32.DL
4E454C33322E	NEL32.
656C33322E64	el32.d
6C33322E646C	l32.dl
400000C02E72	@?.r
300000006000	0‘
600000000100	‘
00004B45524E	KERN
6B65726E656C	kernel
004C6F61644C	LoadL
322E646C6C00	2.dll
33322E646C6C	32.dll
00004C6F6164	Load
657450726F63	etProc
000047657450	GetP
000040000010	@

Table 12 Subset of 74 6-grams selected by Elastic-Net on group A. Some non-printable and repeated ASCII 6-grams removed for legibility

6-gram	ASCII decoding
647265737300	dress
322E646C6C00	2.dll
617279410000	aryA
627261727941	braryA
726172794100	raryA
000000002E72	.r
0000002E7273	.rs
00002E727372	.rsr
002E72737263	.rsrc
010006000100	
03004D005500	MU
004D00550049	MUI
0000004D0055	MU
550049000000	UI
000040000010	@
400000100000	@
004000001000	@
006500730063	esc
006F00700079	opy
006500670061	ega
007900720069	yri
007000790072	pyr
006E002E0020	n.␣
73006F006600	sof
740020004300	t␣C
0000004D0069	Mi
690063007200	icr
6F0073006F00	oso
160001004300	C
001600010043	C
660074002000	ft␣
0072006F0073	ros
004D00690063	Mic
000000004D00	M
0000004C0016	L
000000A90020	?␣
0074000000A9	t?
0000A9002000	?␣
0020004D0069	␣Mi
440000000100	D
000001005600	V
010056006100	Va
000100560061	Va
6F0072006100	ora
43006F007200	Cor
0043006F0072	Cor
6F0072007000	orp
720070006F00	rpo
20004D006900	␣Mi

Table 13 Subset of 64 4-grams selected by Lasso on group A. Some non-printable and repeated ASCII 4-grams removed for legibility

4-gram	ASCII decoding
436C6F73	Clos
74610000	ta
6C654100	leA
00457869	Exi
4C696272	Libr
47504144	GPAD
322E646C	2.dl
8A526963	Ric
61727941	aryA
50726F63	Proc
00002E72	.r
4D006900	Mi
00200043	␣C
00A90020	©␣
006E0037	n7
55004900	UI
0073006F	so
AE002000	®␣

Table 14 Subset of 80 4-grams selected by Lasso on group B. Some non-printable and repeated ASCII 4-grams removed for legibility

4-gram	ASCII decoding
74616D70	tamp
69006700	ig
72617469	rati
696D6520	ime␣
00650064	ed
68616E67	hang
6E61626C	nabl
4D006500	Me
6F667420	oft␣
636F6D6D	comm
5F6F6E65	_one
6C696461	lida
69646174	idat
6E74696D	ntim
2E706462	.pdb
6F647563	oduc
0041006C	Al
6D616C6C	mall
72616D65	rame
7465726D	term
696F6E73	ions
3A2F2F73	://s
5554462D	UTF-
3D227265	="re
6C3D2272	l="r
72654164	reAd
6541646D	eAdm
2E637274	.crt
3C2F7365	</se
666F2078	fo␣x
2F747275	/tru
2F726571	/req