A topic modeling-based approach to executable file malware detection

Waleed Hilal*[a], Connor Wilkinson[a], Naseem Alsadi[a], Onur Surucu[a], Alessandro Giuliano[a], Stephen A. Gadsden[a], John Yawney[b]
[a]McMaster University, 1280 Main St. W, Hamilton, ON, CA L8S 4L8;
[b]Adastra Corporation, 200 Bay St., Toronto, ON, CA M5J 2J2

ABSTRACT

Malware is a term that refers to any malicious software used to harm or exploit a device, service, or network. The presence of malware in a system can disrupt operations and the availability of information in networks while also jeopardizing the integrity and confidentiality of such information, which poses a grave issue for sensitive and critical operations. Traditional approaches to malware detection often used by antivirus software are not robust in detecting previously unseen malware. As a result, they can often be circumvented by finding and exploiting vulnerabilities of the detection system. This study involves using natural language processing techniques, considering the recent advancements made in the field in recent years, to analyze the strings present in the executable files of malware. Specifically, we propose a topic modeling-based approach whereby the strings of a malware's executable file are treated as a language abstraction to extract relevant topics, which can then be used to improve a classifier's detection performance. Finally, through experiments using a publicly available dataset, the proposed approach is demonstrated to be superior in performance to traditional techniques in its detection ability, specifically in terms of performance measures such as precision and accuracy.

Keywords: Malware detection, anomaly detection, natural language processing, machine learning, security, feature extraction, topic modelling

1. INTRODUCTION

Malicious entities generally use targeted attacks on individuals and organizations to acquire sensitive information or cause harm by disrupting operations [1]. These targeted attacks usually involve delivering harmful executable files to the victim's device or network through various channels such as spam e-mails, compromised websites, or bundled with other software. Upon execution of the harmful files, the system becomes infected with what is known as malware, which the attacker may use to compromise a system's security and allow them access to its contents and functionality.

Antivirus programs use a combination of different detection methods, yet, they have been demonstrated to be growing increasingly ineffective in detecting and protecting against malware [2]. One type of antivirus technology involves static analysis or signature-based approaches, whereby each signature is a set of manually crafted rules used to identify small families of malware [3]. Simply put, this approach involves pattern matching to look for similarities between the file in question and any already identified instances of malware. However, these rules are generally specific to each family of malware and thus fail to recognize new types of malware, even if they have the same functionality. This approach is considered insufficient as millions of new malware samples are found daily.

Instead, the current widespread approach for malware analysis is through a sandbox environment, otherwise known as dynamic analysis [4]. In dynamic analysis, suspicious files are executed in isolated, controlled environments known as sandboxes, such as a virtual machine, and malware features such as the APIs called, instructions executed, IP addresses accessed, and others are monitored [3]. Depending on the behaviour and output of the file, it is then either classified as malware or determined to be safe. Although considered more robust than static analysis, some deficiencies are still associated with the dynamic approach. For example, some malware can detect when they are being analyzed and alter or obfuscate their behaviour to evade discovery [5] [6].

Furthermore, even when this type of evasive behaviour is not displayed, dynamic analysis suffers from the fact that the quarantined environment in which the analysis occurs may not accurately reflect the environment targeted by the malware. Consequently, this results in non-insignificant discrepancies between the data collected and actual environments [7]. Finally, dynamic analysis is considered a computationally intensive task that does not scale efficiently to handle increasing amounts of files for malware analysis.

Disruptive Technologies in Information Sciences VI, edited by Misty Blowers,
Russell D. Hall, Venkateswara R. Dasari, Proc. of SPIE Vol. 12117,
1211708 · © 2022 SPIE · 0277-786X · doi: 10.1117/12.2619033

In this study, we propose a topic modeling-based approach to malware detection, which addresses the need for improved rapid detection of malware in a mass number of files. Specifically, the proposed approach examines executable files' operational code (OpCode) and considers their instructions as sequences of strings. From there, a natural language processing (NLP) approach is taken, treating the sequence of strings from the OpCode instructions as a language. Relevant topics are then extracted from the OpCode and augmented into a classifier's dataset to observe whether the proposed engineered features result in detection improvements in a classifier. Section 2 provides a brief background on the models and algorithms involved in our study. Then, in Section 3, we discuss the dataset involved and its source and provide a detailed walkthrough of the methodology behind the approach proposed in this study. Our results are then presented in Section 4, along with a commentary and discussion of the relevant findings, followed by concluding remarks and suggestions for future research in Section 5.

2. BACKGROUND

2.1 Latent Dirichlet allocation

One of the most popular topic modelling methods is latent Dirichlet allocation (LDA), a generative probabilistic model of a corpus first introduced by Blei *et al.* in [8]. The purpose of LDA is to reveal hidden semantic structures in a collection of textual documents. The idea is that each document exhibits a mixture of latent topics, where each topic z receives a weight $\theta_z^{(d)}$ in document d, and each topic is represented by a probability distribution over a finite set of words, with each word w having probability $\phi_w^{(z)}$ in topic z. It is assumed by the generative model that documents are produced by independently sampling a topic z for each word from $\theta^{(d)}$, and subsequently, independently sampling the word from $\phi^{(z)}$.

The assumption of independence comes from the fact that a document is treated as a 'bag-of-words'; therefore, word ordering is irrelevant to the model. Furthermore, symmetric Dirichlet priors are placed such that $\theta^{(d)} \sim \text{Dirichlet}(\alpha)$, and $\phi^{(z)} \sim \text{Dirichlet}(\beta)$, where α and β are hyperparameters that affect the relative sparsity of these distributions. Thus, the complete model of probabilities can be represented as [9]:

$$w_i | z_i, \phi^{(z_i)} \sim \text{Discrete}\left(\phi^{(z_i)}\right), \qquad i =, 1, \dots, N \tag{2.1}$$

$$\phi^{(z)} \sim \text{Dirichlet}(\beta), \qquad z =, 1, \dots, T \tag{2.2}$$

$$z_i | \theta^{(d_i)} \sim \text{Discrete}\left(\theta^{(d_i)}\right), \qquad i = 1, \dots, N \tag{2.3}$$

$$\theta^{(d)} \sim \text{Dirichlet}(\alpha), \qquad d = 1, \dots, D \tag{2.4}$$

where N represents the total number of words in the collection of documents, T represents the number of topics, D represents the number of documents, while d_i and z_i respectively represent the document and topic of the ith word, w_i. With this model, the goal of inference is to identify the values of ϕ and θ , given a collection of documents represented by a sequence of N words $\mathbf{w}_N = (w_1, \dots, w_N)$. However, the estimation is complicated by the latent variables $\mathbf{z}_N = (z_1, \dots, z_N)$.

Many different algorithms have been proposed to solve this problem, such as the variational expectation-maximization algorithm in [8] and the expectation-propagation algorithm in [10]. Griffiths and Steyvers also proposed a collapsed Gibbs sampling algorithm where ϕ and θ are analytically integrated out of the model such that samples are collected directly from $P(\mathbf{z}_N|\mathbf{w}_N)$ [11]. By using conjugate Dirichlet priors on ϕ and θ, the analytic integration becomes straightforward, and it is thus easy to recover the posterior distributions on ϕ and θ given $\mathbf{z}_N$ and $\mathbf{w}_N$. This means that it is sufficient to estimate ϕ and θ from a set of samples from $P(\mathbf{z}_N|\mathbf{w}_N)$.

2.2 Logistic Regression

A simple and well-known algorithm often used for classification is logistic regression, an extension of the linear regression model used for continuous value prediction. However, in logistic regression, the possible outcomes in a classification problem are modelled as probabilities of belonging to each class. The hypothesis, h_θ of the logistic regression model can be formulated as:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.5}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

where x represents the input data, θ^T represents the transpose of the vectorized model parameters, and $g(z)$ represents the sigmoid or logistic function. Given this logistic regression model, the parameters θ can be solved via maximum likelihood estimation.

2.3 Support Vector Machines

Support vector machines (SVM) are another classification model that maps the input space into a higher dimensional feature space to find an optimal separating hyperplane. They can achieve this without introducing any further computational complexity [12]. The ability of SVMs to work with many features has made them attractive for detection tasks involving a highly imbalanced data set due to their ability to extract relevant and important features. Kernel functions are the mechanism behind how this is made possible, thanks to the nature of SVMs possessing the property of kernel representation. Another property of SVMs is margin optimization, which results from the criteria that select the hyperplane, which minimizes overfitting by maximizing the margin of separation from the two classes.

With an input space X and a higher dimensional space H, a kernel function k is defined as $k(x_1, x_2) = \langle(\Phi(x_1), \Phi(x_2)\rangle$ where $\Phi\text{: X} \rightarrow \text{H}$ is a mapping transforming the input space into a higher-dimensional space. The choice of kernel function depends on the task's application and nature, and the most frequently used are polynomial functions, radial basis functions, and linear functions.

2.4 XGBoost

Following the successful extraction of relevant features using LDA and their augmentation into the dataset, there exist many choices of classifiers that can be used to discern between the malignant and benign classes (e.g. malware and non-malware). In this study, we propose using the popular eXtreme Gradient Boosting (XGBoost) classification model, which is based on the gradient-boosted decision tree algorithm (GBDT) [13]. In addition, XGBoost can be considered state-of-the-art in classification models, performing exceptionally well in detection accuracy while effectively using computational resources to be able to handle billions of samples with far fewer resources than traditional approaches.

The XGBoost model is composed of multiple regression trees, whereby the final output is the consequence of the additive combination of the decision results of all subtrees. This is known as an ensemble approach, where with a high number of individually weak but complementary classifiers, the resultant is a robust estimator. The term boosting refers to the nature in which new models are added to the ensemble sequentially, where at each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far [14]. Thus with XGBoost, the principle idea is to construct the new base-learners to be maximally correlated with the negative gradient of the loss function associated with the whole ensemble [14].

3. METHODOLOGY

This section discusses the design of our proposed solution, including how the data is acquired and processed, the feature engineering techniques used, and an overview of the classification models used to evaluate the proposed feature engineering technique. The assembly language data was obtained from a prior experiment [15], wherein ARM assembly code (or OpCode) instructions were extracted from ARM-based machines while running both malware and non-malware (or benign) programs. This OpCode data was processed to prepare it for ingestion into the three previously mentioned classification algorithms: a logistic regression model, an SVM model, and a gradient boosted decision tree model. In addition, several common feature engineering stages were utilized in the data preparation step, primarily using frequency analysis, aggregation techniques and the proposed topical feature created using LDA. Finally, the three classification models were trained and tested both with and without the proposed topical feature. This approach allowed for an accurate comparison of the effect of the proposed topic-modelling approach and its viability, as will be discussed in later sections of this study.

3.1 Obtaining the Assembly Language Data

Firstly, we had to obtain samples of assembly code from different machines running malware and benign programs. To do this, one can utilize a method similar to that discussed by Bragen in [16] and Parildi in [17], wherein executable programs containing malware were run upon different machines' operating systems. The machine's resulting OpCode can then be read and stored throughout the program's execution. To safely obtain such executable programs, it is recommended to use public repositories of malware such as VirusShare [18] (for PC malware) or CICMalDroid [19] (for mobile malware), rather than using a honey-pot approach yourself, due to the readiness of the existing repositories. Due to its complication and our lack of additional machines to safely run benign and malware programs, a previously acquired dataset of OpCode was obtained from the University of Guelph's CyberScienceLab [15]. This dataset consisted of 512 samples of OpCode instructions, with 244 examples of malware instructions and 268 examples of benign instructions. Though this was not an optimal quantity of data, the relatively even division of data between the two labelled classes allowed us to perform a non-skewed analysis with all 512 samples of data. Furthermore, more OpCode data can be gathered using the proposed methodology outlined above, and experiments should be conducted with a greater quantity of OpCode data to validate our results further.

3.2 Primary Data Analysis

It is important to understand the data used for this research, including identifying the distribution of malware and benign labels and understanding the frequency of instructions present in all the programs. As previously mentioned, the distribution between labels was quite even in our dataset, as shown below in Table I.

Table I. Distribution of class labels in the dataset explored

	Benign Labels (%)	Malware Labels (%)
Label Distribution	52.34	47.65

Next, the frequencies of each instruction in each program or file yielded information regarding which instructions were most used for all instruction sets and instructions found in either the malware or the benign files. Firstly, however, the OpCode data required some cleaning, as many instructions had numbers next to the actions, indicating the order in which the action was occurring. For example, there were many 'AD' instructions, though each was given as 'ADD.X', where X was an indicator of the ADD action's order. To address this, only alphabetic characters were retained to obtain the proper frequencies of each action. After this cleaning process, it was determined that 356 instructions existed in the dataset, and the frequencies for each of the instructions were calculated across all instruction sets. These frequencies are shown in Figures Figure 1, Figure 2 and Figure 3 below.

These frequencies provide important information regarding the contents of the malware and benign instructions. Specifically, we can note that certain instructions occur rather often, regardless of if the instructions are from benign or malware programs, such as 'mov', 'ldr', and 'add'. More importantly, we note that the instructions 'bl' and 'sub' are slightly more prominent in malware instructions than in benign instructions, which could eventually be an important feature when distinguishing between the two classes.

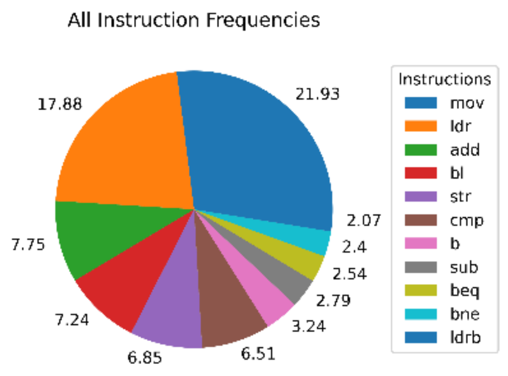

Figure 1. Frequency distribution of each instruction when considering all the possible instructions encountered in both malware and benign cases.

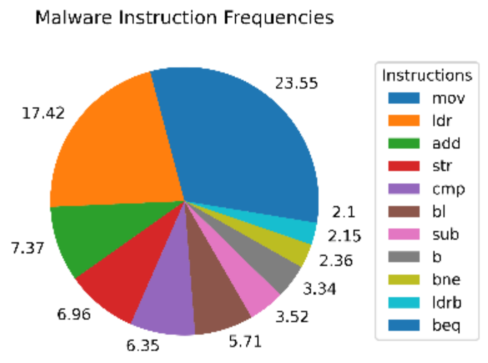

Figure 2: Frequency distribution of each instruction when considering only the malware cases.

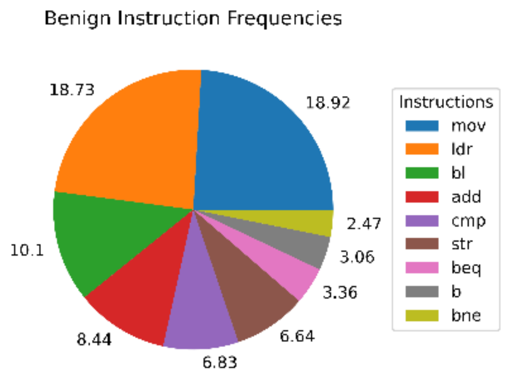

Figure 3: Frequency distribution of each instruction when considering only the benign cases.

3.3 Generic Feature Engineering

The next required step was to generate numerical features as an input for our classification models. For natural language processing models, we must typically convert the terms from categorical features to numerical ones to allow for handling by the different classification models. We began by creating generic frequency features commonly engineered when handling categorical text features for simple classification models like the logistic regression model. We first calculated the frequency for which terms showed up in each set of instructions associated with a program or file. These frequencies were different from the frequencies shown in Figures Figure 1, Figure 2 and Figure 3, as here we were checking for the number of times per OpCode set or program file, where a given instruction showed up, rather than analyzing all of the OpCodes together. We could use this new frequency to indicate feature importance across OpCodes. This new frequency also ensures that we retain information about whether the given term repeats across different programs' OpCode rather than potentially showing up many times in a single program's OpCode and biasing the dataset due to a single program. For each instruction, we then added a new feature to indicate the number of times that instruction was used in each instruction set. These features were valuable insights into the occurrences of instructions within different instruction sets and permitted us to calculate some additional important features.

In addition to gathering the term counts per instruction set, we identified and listed the common and uncommon instructions across all instruction sets. Common instructions were defined as appearing in between 80% - 90% of the OpCode instruction sets, whereas uncommon instructions were defined as appearing in less than 0.5% of the OpCode instruction sets. Instructions appearing in more than 90% of the instruction sets were considered overly abundant and left unused. After creating these two lists, we counted how many of these terms appeared in each instruction set, which then allowed us to create two new features, for the common and uncommon terms, to reflect these counts. This process is then repeated to find common and uncommon instructions for the malware-only and benign-only instruction sets. These four new lists were then used to create four additional features reflecting the degree of intersection between common or uncommon terms and the instruction set itself.

3.4 Topical Feature Engineering

Following the creation of the generic features, we decided to identify 'topics' in each of the instruction sets. To implement this, we used LDA to identify different topics in the instruction code. As previously described, LDA is one of the most popular methods for performing topic modelling. More specifically, it attempts to relate sets of words (or sentences) to one another based on the Dirichlet distribution of words within those sets.

A caveat to the LDA model is that it requires us to specify the number of topics we are looking for. Since this is an unknown, we use coherence measures to identify the best number of topics to use. Coherence measures score topics by measuring semantic similarity between frequency words in the topic. There are multiple coherence measures that we could use, including C_v, UMass or UCI [20]. We decided to use the C_v coherence since it measures the similarity between the top words in a topic, along with the cosine similarity and normalized pointwise mutual information (NPMI) between sentences in topics [20]. Other measures, including UMass or UCI, more highly weigh the frequencies of words in the topics rather than comparing the relative structures of the sentences [21]. These are good measures when the size of the corpus is extensive. However, due to our smaller dataset, we would rather put more weight on the similarities between instruction sets than the number of uses of the individual words.

Additionally, these sets have no 'stop words' or less important instructions. In contrast, when analyzing a language, certain frequently used words can often be considered irrelevant (such as 'and' or 'the'). In such cases, one should provide more weight to the occurrence of the terms across all documents to identify the irrelevant ones, and there, UMass or UCI may be a better measure of coherence. Due to the size of our corpus, we would prefer to weigh the similarities of the semantic shapes of sets over the frequencies of the terms and thus reaffirming our choice to use the C_v coherence measure.

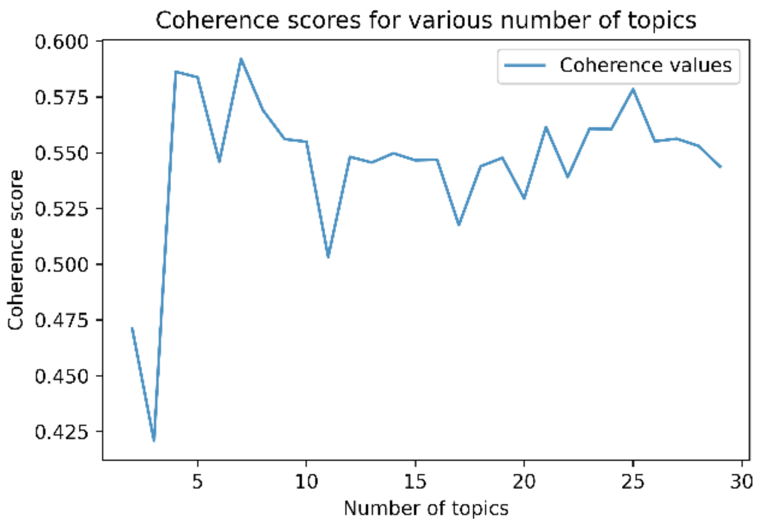

Figure 4. Coherence values associated with different choices for the number of topics that an instruction set can be assigned to.

As shown in Figure 4, we tested a range of topic numbers varying between 2 and 30 and measured the associated coherence values. It was then determined that 7 topics provided the optimal coherence score. Following this, we used the LDA model to identify the seven topics in the OpCode sets and matched each instruction set with their related topic. The resulting distribution of topics is shown below in Figure 5.

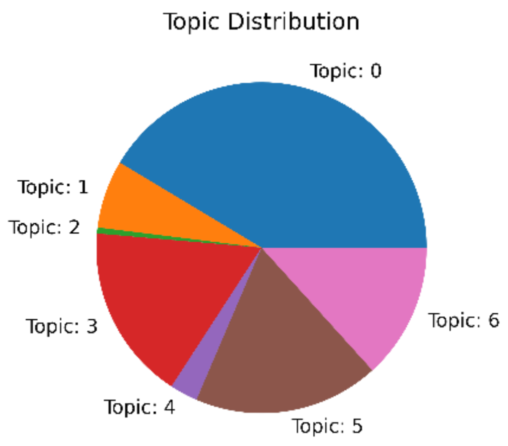

Figure 5. Distribution of topics associated with each instruction set of the analyzed executable files.

Next, we needed to test the correlation of the topics determined to the malware and benign labels since there was no guarantee that the topics found would relate to the labels. As shown in Figure 6, we thus measured the correlation between the malware and benign labels with the topics determined from our previous analysis.

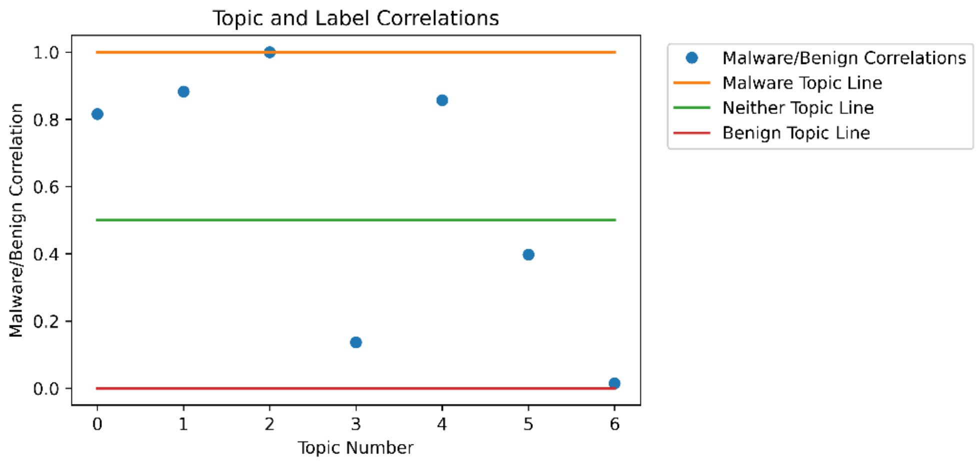

Figure 6. Correlation of each topic with either the benign or malignant class.

As we can see, topics 0, 1, 2 and 4 were highly correlated with malware labels, whereas topics 3 and 6 were highly correlated with benign labels. Topic 5 was the only topic with an instruction set distribution divided between the two labels. Noticing that topics 2 and 4 contained an exceedingly low quantity of instruction sets, we can surmise that topics 0 and 1 will likely be valuable indicators of malware. In contrast, topics 3 and 6 will likely be valuable indicators of benign labels. Subsequently, each instruction set is labelled with its related topic, and the topic feature was added to the training set.

3.5 Classification Models

After creating each of our input features, three classification models were implemented to test the efficacy of our new label feature. These models include a simple logistic regression model, an SVM model, and a gradient boosted decision tree classification model. We chose these three models to show the effectiveness of the additional topic label for various classification models, regardless of complexity. For the logistic regression model and the SVM model, we used the Scikit-learn package in Python, and for the gradient boosted decision tree model, we utilized the XGBoost Python library. Then, using each of the engineered features as an additional input, we created two models per classification model, one with the additional LDA topic feature and one without the proposed additional feature. We utilized the same train and test set for each of the different classification models experimented with, split using a 75% train-test ratio.

4. RESULTS AND DISCUSSION

To measure the performance of each of the different models examined, we use several metrics such as an accuracy score, a precision score and a recall score. The formulas used for each of the accuracy methods are defined below:

$$accuracy \quad = \quad \frac{1}{n}\left(\sum_{i=1}^{n} |\hat{y}_i - y_i|\right) \tag{4.1}$$

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

where, $\hat{y}_i$ is the predicted value, y_i is the true value, TP is the number of true positives predicted, FP is the number of false positives predicted, and FN is the number of false negatives predicted. Note that each of these performance measures is bounded between 0 and 1. The performance results associated with each of the classification models are shown in Table I below.

Table II. Performance of different classification models with and without the proposed topical feature engineering strategy

	Precision		**Recall**		**Accuracy Score**	
	Without Topic Label	*With Topic Label*	*Without Topic Label*	*With Topic Label*	*Without Topic Label*	*With Topic Label*
Logistic Regression	0.67	0.73	0.83	0.77	0.71	0.82
SVM	0.72	0.83	0.83	0.82	0.75	0.84
XGBoost	0.71	0.89	0.98	0.97	0.79	0.89

Reviewing the results in the above table, several observations can be made. Firstly, regardless of the classification model used, a significant increase in precision and accuracy is recorded when using the proposed topic labels compared to the models without the topic labels. For example, in terms of precision, an increase of 6, 11 and 18 percent was demonstrated for the logistic regression, SVM and XGBoost models, respectively. Similarly, an increase in accuracy of 11, 9 and 10 percent was achieved, respectively, for the same models. Thus, one can infer that our proposed topic-based features, when augmented into the training set for a classification model, result in a significant decrease in the number of false positives.

However, it is interesting to note the cost associated with this increase in precision and accuracy. Specifically, we observe a slight decrease in recall across all three classification models with the proposed topic labels – meaning a greater number of false negatives. This effect is most noticeable in the logistic regression model, which suffers from a 6 percent drop in recall, compared to just 1 percent for both SVM and XGBoost. This can be attributed to the fact that logistic regression is a much simpler and less robust model than SVM and ensemble methods like XGBoost. Despite the small drop in recall for the SVM and XGBoost model, the gain in precision and accuracy is significant enough to consider our proposed approach superior in performance.

One more important inference can be made by noticing the significant drop in recall associated with the logistic regression model using our proposed topical features. For instance, the cost of false negatives in an application such as malware detection is considered far more harmful than the cost of false positives. Seeing as the logistic regression model suffered from a significant decrease in recall, careful considerations must be made in terms of the choice of a model when considering the use of our approach. If chosen carefully, such as in the case of the XGBoost and SVM models, the drop in recall can be minimized to the point that the benefits associated with the increase in precision and recall by adding topical features are considered worthwhile.

5. CONCLUSION

In this study, a topic modelling-based feature engineering approach was proposed to improve the detection accuracy of classification models applied to the task of malware detection. Our strategy inspects the assembly language or OpCode of executable files, regarding the instructions as a language from which further features are created. Latent Dirichlet allocation was also used to categorize each executable file's instructions into a topic, forming another proposed feature augmented into the training set. With our approach, when these features are augmented into the training set of a classification model, a significant increase in accuracy and precision can be observed regardless of the classification model used. However, this came at the cost of a slight decrease in recall, where it was then shown that this drop in recall could be minimized to as little as one percent with the proper considerations and choice of a classification model. Overall, our model addresses the need for a more effective, accurate, and rapid detection technique than existing methods in practice. Some limitations are involved with our study; however, we believe it is a good starting point for evaluating the practical performance of our approach. Namely, we note a dearth of data available for this application, which poses a significant bottleneck for developing models and algorithms in this area of study and their resultant performance. Future work involves exploring more advanced topic modelling techniques to address the drop in recall associated with our approach. This includes investigating models such as long short-term memory (LSTM) and the transformer model to take advantage of their advanced sequence-modelling abilities, which are inherently useful for NLP tasks and applications.

REFERENCES

[1] W. Hilal, S. A. Gadsden and J. Yawney, "Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances," *Expert Systems with Applications,* vol. 193, p. 116249, 2022.

[2] M. Al-Asli and T. A. Ghaleb, "Review of Signature-based Techniques in Antivirus Products," in *2019 International Conference on Computer and Information Sciences (ICCIS)*, Sakaka, Saudi Arabia, 2019.

[3] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro and C. Nicholas, "Malware Detection by Eating a Whole EXE," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2018.

[4] M. Lindorfer, C. Kolbitsch and P. M. Comparetti, "Detecting Environment-Sensitive Malware," in *14th International Symposium, RAID*, Menlo Park, CA, 2011.

[5] M. Carpenter, T. Liston and E. Skoudis, "Hiding Virtualization from Attackers and Malware," *IEEE Security & Privacy,* vol. 5, no. 3, pp. 62-65, 2007.

[6] T. Raffetseder, C. Kruegel and E. Kirda, "Detecting System Emulators," in *International Conference on Information Security*, Valparaíso, Chile, 2007.

[7] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos and M. van Steen, "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook," in *IEEE Symposium on Security and Privacy*, San Francisco, CA, 2012.

[8] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research,* vol. 3, no. 4-5, pp. 993-1022, 2003.

[9] K. Canini, L. Shi and T. Griffiths, "Online Inference of Topics with Latent Dirichlet Allocation," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, Clearwater Beach, FL, 2009.

[10] T. P. Minka and J. Lafferty, "Expectation-Propagation for the Generative Aspect Model," in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI2002)*, Edmonton, AB, 2002.

[11] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences,* vol. 101, no. suppl 1, pp. 5228-5235, 2004.

[12] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995.

[13] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2016.

[14] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics,* vol. 7, p. 21, 2013.

[15] CyberScienceLab, "github.com," University of Guelph, 2021. [Online]. Available: https://github.com/CyberScienceLab/Our-Datasets/tree/master/IoT/OpCode/OpCode. [Accessed 20 January 2022].

[16] S. R. Bragen, Malware detection through opcode sequence analysis using machine learning, Master's Thesis, Gjvik: Gjvik University College, 2015.

[17] E. S. Parildi, Deep Learning Aided Runtime Opcode Based Malware Detection, Master's Thesis, Toronto: University of Toronto, 2019.

[18] "VirusShare," [Online]. Available: https://virusshare.com/. [Accessed 20 January 2022].

[19] U. o. N. Brunswick, "CICMalDroid 2020," University of New Brunswick, [Online]. Available: https://www.unb.ca/cic/datasets/maldroid-2020.html. [Accessed 19 January 2020].

[20] S. Syed and M. Spruit, "Full-Text or Abstract? Examining Topic Coherence Scores Using Latent Dirichlet Allocation," in *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Tokyo, Japan, 2017.

[21] M. Röder, A. Both and A. Hinneburg, "Exploring the Space of Topic Coherence Measures," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, Shanghai, China, 2015.