APPROVAL SHEET

Title of Thesis: Random Forest of Tensors

Name of Candidate: Maksim E. Eren
Master of Science in Computer Science
May 2022

Thesis and Abstract Approved: ______________________

Charles K. Nicholas, Ph.D.
Professor
Department of Computer Science
and Electrical Engineering

Date Approved: April 23, 2022

Curriculum Vitae

Name: Maksim E. Eren

Degree and date to be conferred: Computer Science, M. S., 2022

Collegiate institutions attended:

- University of Maryland, Baltimore County, Computer Science, M.S., 2022
- University of Maryland, Baltimore County, Computer Science, B.S., 2020
- Montgomery College, Computer Science, A.A., 2018

Professional positions held:

- Graduate Research Assistant, Los Alamos National Laboratory
- Graduate Research Assistant, University of Maryland, Baltimore County
- Undergraduate Teaching Fellow, University of Maryland, Baltimore County
- Undergraduate Research Assistant, Los Alamos National Laboratory
- Undergraduate Research Assistant, Cyber Pack Ventures
- Information Security Intern, Montgomery County Government EISO

Conference Publications:

1. **Eren, M.E.**, Solovyev, N., Hamer, C., McDonald, R., Alexandrov, B., and Nicholas, C.. COVID-19 Multidimensional Kaggle Literature Organization. In DocEng '21: 21th ACM Symposium on Document Engineering, Aug.

24-27, 2021, Virtual Event, Limerick, Ireland. ACM, New York, NY, USA, 4 pages. DOI: 10.1145/3469096.3474927

2. **Eren, M.E.**, Moore, J.S., and Boian, A.S.. Multi-Dimensional Anomalous Entity Detection via Poisson Tensor Factorization. In ISI '20: Proceedings of the 13th IEEE International Conference on Intelligence and Security Informatics, Nov. 9-10, 2020, Virtual Event, USA., 6 pages. DOI: 10.1109/ISI49825.2020.9280524

3. **Eren, M.E.**, Solovyev, N., Raff, E., Nicholas, C., and Johnson, B.. COVID-19 Kaggle Literature Organization. In DocEng '20: 20th ACM Symposium on Document Engineering, Sep. 29 - Oct. 2, 2020, Virtual Event, CA, USA. ACM, New York, NY, USA, 4 pages. DOI: 10.1145/3395027.3419591

Journal Publications:

1. **Eren, M.E.**, Moore, J.S., Skau, E.W., Bhattarai, M., Moore, E.A, and Alexandrov, B.. 2022. General-Purpose Unsupervised Cyber Anomaly Detection via Non-Negative Tensor Factorization. Digital Threats: Research and Practice, 28 pages. DOI: 10.1145/3519602

2. Golaszewski, E., Sherman , A.T., [et al, including **Eren, M.E.**]. 2020. Project-based learning continues to inspire cybersecurity students: the 2018-2019 SFS research studies at UMBC. Association for Computing Machinery. ACM, New York, NY, USA, 9 pages. DOI: 10.1145/3386363

Posters & Abstracts:

1. Bhandary, P., Adetunji, I., Kiendrebeogo, A., Vieson, C., Joyce, R.J., **Eren, M.E.**, and Nicholas, C.. Malware Antivirus Scan Pattern Mining via Tensor Decomposition. Poster session to be presented at MTEM '22: Malware Technical Exchange Meeting.
2. Liu, R., **Eren, M.E.**, and Nicholas, C.. Can Feature Selection Help Quantum Machine Learning for Malware Detection?. Poster session to be presented at MTEM '22: Malware Technical Exchange Meeting.
3. **Eren, M.E.**, Nicholas, C., McDonald, R., and Hamer, C.. Random Forest of Tensors. MTEM ;21: Malware Technical Exchange Meeting, July 13-15, 2021, Sandia National Laboratories, Virtual Event, USA.
4. Boutsikas, J., **Eren, M.E.**, Varga, C., Raff, E., Matuszek, M., and Nicholas, C.. Evading Malware Classifiers via Monte Carlo Mutant Feature Discovery. MTEM '21: Malware Technical Exchange Meeting, July 13-15, 2021, Sandia National Laboratories, Virtual Event, USA.

Papers Under Review:

1. **Eren, M.E.**, Bhattarai, M., Joyce, R.J., Raff, E., Nicholas, C., and Alexandrov, B.. Semi-supervised Classification of Malware Families via Hierarchical Non-negative Matrix Factorization with Automatic Model Determination. Under review at ACM Transactions on Privacy and Security (TOPS) journal.

ABSTRACT

Title of thesis: Random Forest of Tensors
MASTER'S THESIS

Maksim E. Eren, Master of Science, 2022

Thesis directed by: Professor Charles Nicholas
Department of Computer Science and
Electrical Engineering

Tensor decomposition is a powerful unsupervised Machine Learning method that enables the modeling of multi-dimensional data, including malware data. This thesis introduces a novel ensemble semi-supervised classification algorithm, named Random Forest of Tensors (RFoT), that utilizes tensor decomposition to extract the complex and multi-faceted latent patterns from data. Our hybrid model leverages the strength of multi-dimensional analysis combined with clustering to capture the sample groupings in the latent components, whose combinations distinguish malware and benign-ware. The patterns extracted from a malware data with tensor decomposition depend upon the configuration of the tensor such as dimension, entry, and rank selection. To capture the unique perspectives of different tensor configurations, we employ the *"wisdom of crowds"* philosophy and make use of decisions made by the majority of a randomly generated ensemble of tensors with varying dimensions, entries, and ranks. We show the capabilities of RFoT when classifying Windows Portable Executable (PE) malware and benign-ware.

Random Forest of Tensors

by

Maksim E. Eren

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science
2022

Advisory Committee:
Dr. Charles Nicholas, Chair/Advisor (University of Maryland Baltimore County)
Dr. Edward Raff (Booz Allen Hamilton, University of Maryland Baltimore County)
Dr. Roberto Yus (University of Maryland Baltimore County)
Dr. Boian S. Alexandrov (Los Alamos National Laboratory)

© Copyright by
Maksim E. Eren
2022

Acknowledgments

I would like to extend my highest gratitude to Dr. Charles Nicholas who had been a great mentor, advisor, teacher, and supporter who fostered my growth as a student and researcher throughout my education at University of Maryland, Baltimore Country (UMBC). I would also like to thank Dr. Edward Raff, from whom I had the great privilege of receiving mentorship in Machine Learning and research in general. I furthermore would like to extend my highest appreciation to my mentor, Dr. Boian Alexandrov. It has been an extraordinary privilege to learn tensor decomposition and ML methods from Dr. Alexandrov, who always pushed me to improve, learn, explore, and discover together while always supporting my success as a researcher and student. I would also like to express my highest gratitude to Juston S. Moore, who introduced me to tensor decomposition, spent countless hours teaching me the foundations of tensors and anomaly detection, and constantly supported my growth as a professional. I would also like to thank Dr. Roberto Yus for all his help and support in the preparation of my thesis. Additionally, I would like to thank Dr. Manish Bhattarai and my other colleagues from Los Alamos National Laboratory (LANL) for all of their mentorship and continuous support. Furthermore, I would like to thank all the students from the UMBC DREAM Lab for all the collaboration and brainstorming. It had been a great experience and privilege to work alongside the brightest and most motivated peers I have ever met. I would also like to thank Laboratory for Physical Sciences for their support. Finally, I would like to thank my family and friends who have always supported me.

I especially thank Joe Aurelio, Ryan Barron, Sina Sontowski, and Mark Mohades for their valuable feedback and suggestions.

Contents

List of Tables vi

List of Figures vii

List of Abbreviations viii

1 Introduction 1
- 1.1 Motivation for Automated Malware Detection . . . 1
- 1.2 Motivation on Using Tensor Decomposition for Malware Detection and Thesis Contributions . . . 3
- 1.3 Prior Publication Notice . . . 5
- 1.4 User-friendly Python Library for RFoT . . . 6
- 1.5 Thesis Outline . . . 6

2 Related Work 8
- 2.1 Machine Learning Based Malware Classification . . . 8
- 2.2 Tensor Decomposition and Cybersecurity . . . 12

3 RFoT Methodology 16
- 3.1 Tensor Decomposition Notation . . . 16
- 3.2 Clustering Specimens Over the Latent Components . . . 22
  - 3.2.1 Malware Patterns in the Latent Factors . . . 22
  - 3.2.2 Capturing the Latent Patterns via Clustering . . . 25
- 3.3 Ensemble of Random Tensor Configurations . . . 26
  - 3.3.1 Notation for an Ensemble of Tensors . . . 26
  - 3.3.2 Random Tensor Configuration Sampling . . . 28
  - 3.3.3 Feature to Tensor Dimension Mapping and Tensor Entry . . . 30
- 3.4 Semi-supervised Classification with RFoT . . . 32
  - 3.4.1 Semi-supervised Voting Using the Clusters . . . 32
  - 3.4.2 Cluster Uniformity Score . . . 34
- 3.5 Putting it Together: RFoT Algorithm . . . 35

4 Experiments 38
- 4.1 EMBER-2018 Dataset and Experiment Setup . . . 38
- 4.2 Performance Evaluation Metrics . . . 40
  - 4.2.1 Precision, Recall, and F1 Measure . . . 40

4.2.2 Execution Time and Abstaining Predictions 41
4.3 RFoT Hyper-parameter Performance Analysis 41
4.3.1 Bin Scale 42
4.3.2 Cluster Uniformity Threshold 44
4.3.3 Maximum and Minimum Number of Dimensions 46
4.3.4 Tensor Rank 48
4.3.5 Number of Estimators (Random Tensors) 50
4.4 Baseline Comparisons 52
4.4.1 Labeled Malware Data Scarcity Experiment 54

5 Future Work and Conclusions 57
5.1 Future Work 57
5.2 Conclusions 58

Bibliography 59

List of Tables

3.1 Summary of the notation styles used in the thesis. 16

4.1 Baseline comparisons 52

List of Figures

3.1 Illustration of CPD on a 3-dimensional tensor 19
3.2 Clean malware and benign-ware clusters found by tensor decomposition 22
3.3 Clean malware clusters and noisy benign-ware clusters found by tensor decomposition 22
3.4 Tensor decomposition placing benign samples into a single latent factor 24
3.5 Tensor decomposition placing malware specimens into a single latent factor 24
3.6 Example of 4 numerical feature values being mapped to 5 bins to form a tensor dimension. 30
3.7 Possible cases of clusters that can be seen 32
3.8 RFoT methodology overview 36

4.1 Abstaining percentage is shown for different values of the Bin Scale. . 42
4.2 F1 score is shown for different values of the Bin Scale. 42
4.3 Abstaining percentage is shown for different values of the Cluster Uniformity Threshold. 44
4.4 F1 score is shown for different values of the Cluster Uniformity Threshold. 45
4.5 Abstaining percentage is shown for different values of the Max Dimensions parameter. 46
4.6 F1 score is shown for different values of the Max Dimensions parameter. 46
4.7 Abstaining percentage is shown for different values of the Min Dimensions parameter. 47
4.8 F1 score is shown for different values of the Min Dimensions parameter. 47
4.9 Abstaining percentage is shown for different values of the Rank parameter. 48
4.10 F1 score is shown for different values of the Rank parameter. 48
4.11 Abstaining percentage is shown for different values of the Num Estimators parameter. 50
4.12 F1 score is shown for different values of the Num Estimators parameter. 50
4.13 Abstaining percentage is shown for different values of the unknown fraction. 54
4.14 F1 score is shown for different values of the unknown fraction. 54

List of Abbreviations

- **ML:** Machine Learning
- **AI:** Artificial Intelligence
- **RFoT:** Random Forest of Tensors
- **UMBC:** University of Maryland, Baltimore County
- **CPD:** CANDECOMP/PARAFAC Decomposition
- **CP-ALS:** CANDECOMP/PARAFAC Alternating Least Squares
- **CP-APR:** CANDECOMP/PARAFAC Alternating Poisson Regression
- **IDS:** Incident Detection System
- **PE:** Portable Executable
- **CI:** Confidence Interval
- **KL:** Kullback-Leibler
- **MU:** Multiplicative Update
- **TF-IDF:** Term Frequency - Inverse Document Frequency
- **VEC:** Voting Ensemble Classifier
- **BWMD:** Burrows Wheeler Markov Distance
- **DL:** Dictionary Learning
- **CNN:** Convolutional Neural Network
- **DNN:** Deep Neural Network
- **SGAN:** Semi-supervised Generative Adversarial Network
- **HPC:** High Performance Computing

Chapter 1: Introduction

1.1 Motivation for Automated Malware Detection

Malware is a generic term for any unwanted software with a purpose of stealing personal or confidential information, or causing harm to the system when deployed. Recent cyber reports rank malware as one of the most costly and frequent cyber threats [1]. In general, the yearly cost of malware to organizations is reported to be $2.5 million [1], while the total cost of a single ransomware breach is nearly $4.62 million [2]. At the same time, the quantity of malware in the wild continues to grow rapidly. Approximately 13.5 million new malware specimens and unwanted applications are reported monthly, adding up to a total of nearly 1.3 billion known malware specimens in 2022 [3]. This rapid increase in the quantity of malware is accompanied with the growing sophistication and threat capabilities, making the problem of defending against malware more challenging [4, 5]. The growing capabilities, sophistication, cost, and the quantity of malware combined with the lack of an experienced malware analyst workforce to respond to the immense number of malware attacks drives the need to utilize automated defense systems based on Machine Learning (ML) to combat malware. ML can enable early detection and reduce response times allowing automation to reduce the cost of a security breach by

80% [2]. There are two categories of features used for ML-based malware detection, each with its own advantages and disadvantages.

Dynamic malware analysis-based features are collected at run-time and often include system calls, file system events, network, and process activity. In comparison, static-malware-analysis-based features directly come from the contents of an executable binary such as the Portable Executable (PE) header, strings, code, and raw bytes. The static malware features can be collected from a specimen without a need to execute the binary. Dynamic analysis-based features often provide a more detailed picture of the malware behavior and are less prone to possible obfuscations and packing techniques [6]. However, obtaining dynamic features has several challenges. Dynamic features require executing the malware in a resource-expensive isolated sandbox environment that often result in a slow feature collection process. In addition, some malware is capable of detecting the presence of a sandbox and modifying its behavior [7]. Despite the potential shortcomings, static malware features are still an effective way to detect and characterize malware. We refer the reader to [6] for more details on classical malware analysis. In this study, we focus on static-malware-analysis-based features, specifically Windows PE header features, to present an operationally relevant automated non-real time solution that can be adopted by the blue-teams in an effort to combat malware. More specifically, our proposed malware classification tool is based on tensor decomposition where the malware data is modeled in a multi-dimensional space.

1.2 Motivation on Using Tensor Decomposition for Malware Detection and Thesis Contributions

Tensor decomposition is a powerful unsupervised ML method that can extract multi-faceted latent patterns from complex and large-scale data. Compared to the classical ML methods, which are often black-box, tensor decomposition produces interpretable results making it an effective tool for incident responders who need to verify the alerts generated by automated incident detection systems (IDS). Due to the fact that malware authors continuously generate new malware variants [8], ML models used in malware identification need to be able to detect novel malware. However, many of the popular ML models used in cyber defense are supervised solutions that do not generalize well to new data. At the same time, supervised ML models need a large quantity of labeled data to yield the desired performance when deployed in production systems. This is problematic for the malware domain because labeled malware data is expensive and time-consuming to obtain [8]. Semi-supervised methods, including the algorithm that we propose in this thesis, can help in the areas where the supervised methods fall short as they have the potential of improved generalizability to new data, and the capability to yield good performance with low quantity of labeled data. Despite the potential benefits of semi-supervised solutions, the research community had not widely explored the application of semi-supervised learning to Windows malware detection [8]. We address this gap in malware research field, and introduce a semi-supervised algorithm based on tensor decomposition to classify Windows malware.

Malware data is naturally multi-dimensional; therefore, we can construct a multi-dimensional representation of a malware using tensors and analyze it via tensor decomposition methods. In this work, we introduce an ensemble semi-supervised classification algorithm, named Random Forest of Tensors (RFoT), that utilizes the ability of tensor decomposition to extract meaningful patterns from multi-dimensional malware data. We find that the malware and benign samples form clusters within and among the latent components extracted with tensor decomposition. We capture these groupings using clustering methods and perform semi-supervised class voting for each unknown specimen grouped in a cluster using the known samples from the same cluster. The noisy clusters are removed based on a cluster uniformity threshold that is calculated using the known specimens. The extracted latent patterns depend on the configuration of the tensor, including the tensor dimension, entry, and rank selection. RFoT employs the *"wisdom of crowds"* philosophy, and obtain the final class prediction with max-vote over the votes received from each randomly generated ensemble of tensor configurations with varying dimensions, entries, and ranks.

In this study, we investigate the malware classification performance of RFoT using two tensor decomposition algorithms with distinct characteristics: CANDECOMP/PARAFAC Alternating Least Squares (CP-ALS) [9–11], and CANDECOMP/PARAFAC Alternating Poisson Regression (CP-APR) [12] tensor decomposition. We also compare Mean Shift (MS) [13–16] and Component clustering algorithms that are utilized to capture the patterns from the latent factors. In our experiments, we use Windows Portable Executable (PE) file features of malware and benign-ware from the EMBER-2018 dataset [17]. Specifically, we create 10 random subsets of size 10,000 PE files and apply our experiments in each random subset to show that our results are statistically significant using 95% confidence intervals. We baseline our method against the tuned *XGBoost* [18] and *LightGBM* [19] models, which are used by the prior research to report high classification scores. We also compare our results against *SelfTrain* extended *XGBoost* model which creates a semi-supervised learner [20]. We show that our semi-supervised method outperforms the state-of-the-art supervised models with an F1 score of 0.968 when classifying malware, with a trade-off on reduced number of predicted labels due to the abstaining predictions (i.e. predict *"we do not know what class this is"*). To the best of our knowledge, we are first to formulate a tensor based semi-supervised classifier with an ensemble learning framework to classify Windows malware.

1.3 Prior Publication Notice

RFoT was previously presented in the Malware Technical Exchange Meeting (MTEM) 2021 poster session at the Sandia National Laboratories by Eren, Nicholas, McDonald, and Hamer [21]. In this thesis, we get the chance to take a deeper dive into understanding the capabilities of RFoT when classifying malware and benign files. Specifically, we give a detailed description of our method, present its performance with different hyper-parameter settings, and compare to several baseline models.

1.4 User-friendly Python Library for RFoT

To enable the reproducibility of our results and to provide an operationally-relevant tool, we release RFoT on GitHub[1]. Our Python library follows the well-known Scikit-learn like API to ease the usage of RFoT [16, 22]. RFoT Python library also includes the Python implementation of the CP-ALS algorithm that was originally released in MATLAB Tensor Toolbox [23,24]. For the CP-APR algorithm which was also originally introduced in MATLAB Tensor Toolbox [23, 25], we use the latest Python implementation of the CP-APR algorithm with GPU capability, named *pyCP_APR*, as a dependency in RFoT library [26]. Finally, to reduce the computation time, the RFoT library utilizes embarrassingly parallel decomposition of each random tensor member in the ensemble.

1.5 Thesis Outline

The remaining of the thesis is organized as follows: We provide a summary of related work and recent advances in the ML-based malware analysis field, and the use of tensors in cybersecurity problems in Section 2. Section 3 introduces our RFoT approach by first providing an overview of the tensor notation. Then, we describe how the clustering of malware and benign instances is performed using tensors, and how cluster uniformity score is calculated. Finally, Section 3 describes the random tensor ensemble approach before finalizing the discussion with the semi-supervised formulation. We introduce the dataset and the evaluation metrics in Section 4.

---

[1] RFoT is available at `https://github.com/MaksimEkin/RFoT`

Section 4 also include our RFoT hyper-parameter experiments before showcasing the performance comparison with the baseline models. This section closes the discussion with a case study for labeled malware data scarcity. Finally, before concluding, we provide potential areas for future work in Section 5.

Chapter 2: Related Work

2.1 Machine Learning Based Malware Classification

ML-based automated detection and characterization of malware has been a long-lasting research area. The application of deep learning has shown to be an effective method to classify malware in a supervised manner. Vinayakumar et al. used shallow neural networks to detect malware using PE header features [27]. Fabian et al. also used neural networks, but designed their method to be used in environments with limited compute resources [28]. Compared to these solutions that make use of a selected set of static malware features, Raff et al. introduced a deep learning architecture named MalConv to classify malware directly based on the entire raw byte-sequences of the binary [7]. These methods used static malware analysis-based features to classify malware. Although static malware features are an effective way to identify malice in a file, dynamic malware features can provide additional details about the executable. Vinayakumar et al. take a multi-modular approach using Deep Neural Network (DNN) where the classification is performed using features from static analysis, dynamic analysis, and gray-scale malware image [29].

Several other prior works also performed malware detection based on the images generated from malware [30–34], including work on classifying malware visualizations using an ensemble of random forests [35], and another work on a semi-supervised approach to cluster gray-scale malware images [36]. Wang et al. also used a semi-supervised approach using gray-scale malware images [37]. In their work, byte n-grams translated into fixed-size gray-scale image vectors to be used as a feature during training. Wang et al. perform classification with the gray-scale image vectors using a 1-dimensional Convolutional Neural Network (CNN), a supervised deep learning method, that is strengthened using a semi-supervised Generative Adversarial Network (SGAN).

Prior work had also looked at classical ML methods to classify malware. Kumar et al. showed that XGBoost is an effective model to classify Windows PE malware, after performing low-resource feature selection [38]. Pham et al. also used the EMBER-2018 dataset and showed that the statistical summaries of the original PE features can yield improved detection results using LightGBM which outperforms previously introduced deep learning solution MalConv while requiring fewer resources [39].

The majority of the aforementioned work, which reported excellent malware detection capabilities, is based on supervised learning. However, supervised models often face performance degradation in production when faced with specimens that do not follow the same distribution the model saw during the training time. Qi et al. addressed this problem by integrating unsupervised domain adaptation technique based on adversarial learning into LightGBM for static malware detection [40]. Their approach extends LightGBM to learn domain-invariant features by using the predictions made from each decision tree in the model as a feature space which are then used as an input into the adversarial learning framework.

Another domain that has received a growing interest is the application of ensemble learning to enhance the predictive capabilities of malware classifiers. To this end, several prior research looked into an ensemble approach for Windows malware classification using static features. Atluri showed that several tree-based ensemble models including Random Forests, Bagging Decision Tree Classifier, and Gradient Boosting Classifier among others can be combined into a Voting Ensemble Classifier (VEC) to achieve an improved detection of Windows PE malware [41]. A similar method was explored by Ramadhan et al. who created a voting-based ensemble model using LightGBM, XGBoost, and Logistic Regression [42]. Ramadhan et al. showed that the ensemble of classifiers, each with its own inductive biases, can yield increased accuracy compared to any individual model alone since each member of the ensemble complements others' weaknesses. Ensemble learning framework had also been applied to the deep learning field for malware detection by Dahl et al. [43]. Authors showed an ensemble of neural networks with voting, alongside a novel feature selection method based on dimensionality reduction and random projections, can improve the identification of malware.

While the above prior work for ensemble learning used the voting method, Azeez et al. took a stacking approach with an ensemble of CNNs to form a derived dataset based on the decisions of the base models, which are then used as input to the final prediction layer with ExtraTrees classifier to improve the prediction accuracy [44]. Gupta et al. also took a stacking approach with an ensemble of diverse supervised classifiers [45]. Conversely, Gupta et al. carefully selected the best performing classifiers in the ensemble by first ranking them based on their performance. The best-ranking base models are then used within the stacking ensemble layer to further improve the malware detection capability.

Clustering has also been applied in the framework of ensemble learning to identify malware. Ye et al. introduced a hybrid framework that builds base clusters from an ensemble of clustering algorithms applied separately to TF-IDF build from instruction frequency and instruction n-grams [46]. The introduced method utilizes an ensemble of clustering algorithms with distinct characteristics such as hierarchical clustering and weighted subspace K-medoids to build the base clusters, which are then used to extract the signatures that distinguish malware families. A similar framework, based on an ensemble of hybrid clustering algorithms, had also been proposed by Zhang et al. [47].

Similar to the clustering ensembles, distance metrics had also been used with an ensemble learning structure. Kong et al. use similarity metrics between each pair of malware to label malware families [48]. To obtain the similarity measurements, authors use a set of distinct features such as opcodes, system calls, and file system activity. The set of new distance-based feature vectors, each from a different malware feature, are then used to train an ensemble of classical ML models. The similarity-based approach had also been used by Raff et al. previously, where they introduced the Burrows Wheeler Markov Distance (BWMD), an efficient similarity metric, based on embedding the data to a fixed size vector space, and showed its capabilities when clustering malware [49].

Motivated by the advances and success in prior work for ensemble learning and clustering, in our framework, we adopt ensemble learning and clustering to multi-dimensional analysis via tensor decomposition. In addition, motivated by the potential benefits, we formulate our tensor decomposition-based solution in a semi-supervised methodology. The most similar work to ours, regarding semi-supervised learning, was by Irofti et al. who proposed a semi-supervised solution using Dictionary Learning (DL) to classify Windows PE malware [50]. The proposed framework first trains a dictionary in a supervised fashion, which classifies intermittent new malware, and thereby updates the dictionary with new malware signals in an online unsupervised fashion. DL, which is very similar to matrix factorization, is limited by the information conveyed in a two-dimensional space. Using tensors, we model the data in a higher-dimensional space where each dimension enable inclusion of more details about the nature of the data which leads to extraction of complex and polyperspective information. Due to the complex insights that can be gained from the data using multi-dimensional analysis, recent work had utilized tensors in addressing cybersecurity problems.

2.2 Tensor Decomposition and Cybersecurity

Data relevant to cybersecurity problems are often naturally multi-dimensional making tensors an ideal tool for analyzing cyber data. Several prior works had used tensor decomposition to tackle cybersecurity problems in an unsupervised fashion. CANDECOMP/PARAFAC Decomposition (CPD) decomposition had been a popular tool to identify several types of outliers, or anomalies in cyber data [51,52]. Bruns et al. applied non-negative tensor decompositions, specifically CP-APR algorithm, to identify malicious network activity patterns [53]. The authors utilized the interpretability of the tensor decomposition results, visually analyzed the latent factors, and identified each stage of a cyber breach such as reconnaissance, brute-forcing, data exfiltration, and insider threat. Later, CP-APR with a statistical framework was shown to be an effective method to improve automatic anomaly detection capabilities such as identification of users with compromised credentials, botnet network traffic, spam e-mails, and fraudulent credit card transactions [54,55]. CP-APR had also been used to detect cyber anomalies utilizing High Performance Computing (HPC) resources to perform embarrassingly parallel graph analytics in the latent components [56].

Tucker tensor decomposition had also been a popular algorithm to tackle cybersecurity problems. Kanehara et al. used non-negative Tucker tensor decomposition with thresholding over the latent factors to perform real-time botnet detection on the darknet [57]. Similarly, Tork et al. also used Tucker tensor decomposition on a 3-dimensional tensor to identify telecommunication anomalies [58]. Xie et al. proposed a tensor truncating algorithm for fast low-rank Tucker decomposition of tensors, and used reconstruction error as a metric to detect network anomalies [59]. Sun et al. addressed the network anomaly detection problem with a dynamic Tucker tensor decomposition approach to handling large-scale streaming data [60].

In the classical ML field, ensemble learning has shown to be an effective way to improve the capabilities of any individual model alone. Motivated by this fact, there has been an increasing amount of work in the tensor domain that explored ensemble learning. The ensemble learning approach in multi-dimensional space was first used by Kisil et al. [61]. In their work, an ensemble of latent factors extracted via tensor decomposition was used to train classical ML models to combine the underlying hypothesis from each tensor decomposition and ML model. Similarly, Hou et al. introduced a framework that utilizes tensor decomposition with an ensemble setting to classify Android malware [62]. The authors used the application permissions, API calls, and hardware specifications as features to build the tensor. This tensor is then decomposed with Tensor Filter in a recursive boosting approach to build an ensemble of base models. Hou et al. showed that the ensemble approach, combined with tensor decomposition, significantly improves malware detection capabilities compared to the baseline classical ensemble models. In our previous work, we have also used an ensemble approach and showed that an ensemble of tensors with different ranks can enhance anomaly detection capabilities using statistical p-value fusion techniques [55].

The results produced by the aforementioned prior work heavily rely on the selection of the tensor rank, dimensions, and entry. Building an adequate tensor that would produce good results when decomposed is not a trivial task for several reasons. First, finding the rank of a tensor is NP-hard [11]. At the same time, although it is possible to easily build a tensor that would have an intuitive meaning, selecting the best features for the tensor dimensions and entry could require exhaustive study. This process often involves trial and error where different combinations of features are tested. For instance, several prior works used source IP, destination IP, and temporal information as the tensor dimensions for network traffic data (Netflow). However, there are many other possible features in Netflow data such as bytes transferred, number of packets, source and destination port number, and connection length among others. Meanwhile, we may choose to have the tensor entry to be binary values, counts, or another Netflow feature. Therefore, selecting optimal combinations of features to define a given tensor's dimension and entry from numerous options is an exponential scale problem.

In our proposed methodology introduced in the next chapter, we believe that the ensemble of randomized tensor configurations removes the need to find any single tensor with optimal dimensions and entry. Before introducing our framework, we begin the next chapter by summarizing the tensor notation as a preliminary information.

Chapter 3: RFoT Methodology

3.1 Tensor Decomposition Notation

Tensor decomposition is a powerful data analysis method that can extract complex patterns from data in an unsupervised manner. Using tensors, the data can be represented in a multi-dimensional space where the natural relations among each dimension is explored simultaneously. This higher-dimensional and more complex representation enable understanding, discovery, and interpretation of hidden polyperspective information from data. This section is dedicated to summarize the tensor notations that will become handy when introducing RFoT later in the thesis. We also briefly describe the tensor decomposition algorithms used in RFoT at the end of the section. Summary of the notations used in this thesis is provided in Table 3.1.

Table 3.1: Summary of the notation styles used in the thesis.

Notation	**Description**
x	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\boldsymbol{\mathfrak{X}}$	Tensor
$\mathbf{x}_i$	ith element in the vector
$\mathbf{X}_{i,j}$	Entry located on row i and column j
$\mathbf{X}_{i:}$	ith row
$\mathbf{X}_{:j}$	jth column
$\boldsymbol{\mathfrak{X}}^{(i)}$	Superscript (i) used to identify the ith random tensor
$\boldsymbol{\mathfrak{X}}_{::j}$	jth slice of a tensor
$\circ$	Outer product

Tensors are a higher-order extension of matrices that enable multi-dimensional data representation. A first-order tensor is a vector, an order-2 tensor is called a matrix, and anything with 3 through D dimensions is called a tensor. A D-dimensional tensor is called an order-D tensor, and each dimension is also named the mode of the tensor. For instance, the 1st dimension of a tensor is also called the first mode of the tensor. To build up to an example of how a tensor can be constructed, we can first look at matrices. In a lower-dimensional space, we can represent a bag of words from a set of scientific papers in a matrix $\mathbf{X}$ with dimensions *Documents - Words* and shape $N_{Documents} \times N_{Words}$. In this matrix, an entry $\mathbf{X}_{i,j}$ is the number of times document i used word j. Using tensors, we can create a higher-dimensional representation of the data. Take the example of the scientific paper again with additional information of publication time for each paper. We can represent this data in an order-3 tensor $\boldsymbol{\mathcal{X}}$ with dimensions *Documents - Words - Time* and shape $N_{Documents} \times N_{Words} \times N_{Time}$. An entry $\boldsymbol{\mathcal{X}}_{i,j,k}$ represents the number of times document i used word j at time k (year for instance).

We can generalize this example from a 3-dimensional tensor to a D dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathrm{N_1 \, x \, N_2 x \cdots x \, N_D}}$ such that an entry in the tensor is denoted with $\boldsymbol{\mathcal{X}}_{i_1,i_2,\cdots,i_D}$ where the indexing ranges of each mode are $i_1, i_2, \cdots, i_D \in [0 \leq i_1 < N_1, 0 \leq i_2 < N_2, \cdots, 0 \leq i_D < N_D]$. We borrow the *multi-index* notation, and use $\mathbf{i}$ to denote indexing of D modes such that $\mathbf{i} = i_1, i_2, \cdots, i_D$ and $\boldsymbol{\mathcal{X}}_{\mathbf{i}}$ is the tensor entry [12, 63]. Let $nnz(\boldsymbol{\mathcal{X}})$ be the set of all non-zero entries in the tensor $\boldsymbol{\mathcal{X}}$, and let Ω be the set of all entries including the zeros such that we have a sparse tensor when $nnz(\boldsymbol{\mathcal{X}}) < |\Omega|$ [12]. Here $|\Omega|$ is the size of the tensor which is calculated as follows:

$$|\Omega| = \prod_{d=1}^{D} N_d \tag{3.1}$$

We can use the number of non-zeros and the size of the tensor $\boldsymbol{\mathcal{X}}$ to calculate the sparsity of the tensor as follows:

$$\eta = \frac{nnz(\boldsymbol{\mathcal{X}})}{|\Omega|} \tag{3.2}$$

Tensors built from cyber data are often extremely sparse and large. For instance, tensors build for cyber Netflow can have a sparsity as low as $\eta = 10^{-8}$ [54]. In the Netflow example, the dimensions of the tensor can represent the source and destination device, and the time of the event for network communication. Since many of the devices in a network communicate with a small fraction of the other devices in the same network, the tensors build from Netflow data are often sparse. Similarly, different malware features (such as file size, number of sections, and timestamp) can represent each dimension of the tensor. Each specimen's corresponding feature space will map to a single index along each dimension in the given tensor, similar to the Netflow example, resulting in the tensor being sparse. We can use the sparsity of the tensors to our advantage and avoid storing the entire tensor in the memory, which is often not possible due to the large size of the tensor. To this end, we instead store the tensor as a list of non-zero coordinates and the corresponding list of non-zero values, which is called the Coordinate (COO) format. In COO format, each coordinate denotes the indexing $\mathbf{i}$, and each non-zero value is the entry

$\mathcal{X}_{\mathbf{i}}$.

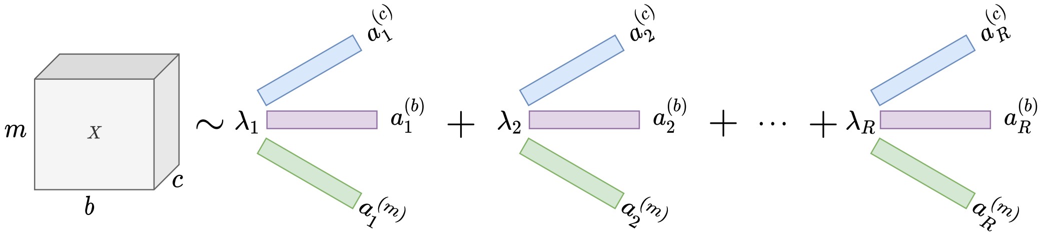

Figure 3.1: Illustration of CPD on a 3-dimensional tensor

The higher-order representation of the data using tensors enables analysis of the information hidden in the data while considering the interactions simultaneously across each dimension allowing extraction of complex and multi-faceted details. The two popular tensor decomposition algorithms used to factorize tensors are Tucker and CANDECOMP/PARAFAC Tensor Decomposition (CPD) [11]. In this thesis, we use CPD to extract the latent patterns from malware data. CPD compresses the D-dimensional tensor $\mathcal{X}$ into lower-dimensional R rank-1 tensors, also called the components, such that the sum of the R rank-1 tensors approximates the original tensor:

$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(D)} \tag{3.3}$$

Here $\circ$ represents the outer product. Latent factors $\mathbf{a}_r^{(d)}$ for each dimension d, where $1 \leq d \leq D$ and rth component where $1 \leq r \leq R$, describe the latent information for the given dimension. Each $\mathbf{a}_r^{(d)}$ normalized to sum up to 1, and weight is absorbed by each λ_r. We also provide an illustration of CPD in Figure 3.1

for a 3-dimensional tensor. The tensor rank R is a hyper-parameter which is chosen by the user. Finding the rank R of a tensor is known to be NP-Hard [11]. We can write CPD in a shorter format with the KRUSKAL notation as follows:

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{M}} \equiv [\![\lambda \,;\; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(D)}]\!] \tag{3.4}$$

Here the KRUSKAL tensor $\boldsymbol{\mathcal{M}}$ is the low-rank approximation of $\boldsymbol{\mathcal{X}}$. Each $\mathbf{A}^{(d)}$ is a matrix of latent factors for dimension d. $\mathbf{A}^{(d)}_{:r}$ is the rth latent factor for dimension d with size N_d such that we can write $\mathbf{A}^{(d)}$ as follows:

$$\mathbf{A}^{(d)} = [\mathbf{a}^{(d)}_1, \mathbf{a}^{(d)}_2, \ldots, \mathbf{a}^{(d)}_R] \tag{3.5}$$

Within each latent factor matrix $\boldsymbol{\mathcal{M}}_{::d-1} = \mathbf{A}^{(d)}$, for dimension d, we can have linearly dependent columns $\mathbf{A}^{(d)}_{:r}$, for each r [11]. However, when all latent factor matrices $\mathbf{A}^{(1,2,\cdots,D)}$ considered together, CPD solutions is almost always unique [11, 64]. The uniqueness of the CPD enable each component to represent distinct events/characteristics of the data. Therefore, the results of CPD gives us interpretable results when we look at each component individually.

In this thesis, we use two popular tensor decomposition algorithms with different properties to heuristically test the malware classification capability of RFoT. The first one is the CANDECOMP/PARAFAC Alternating Least Squares (CP-ALS) tensor decomposition algorithm [9–11]. To fit the tensor $\boldsymbol{\mathcal{X}}$, CP-ALS performs updates, using least squares, by alternating between each latent factor matrix $\mathbf{A}^{(d)}$, while fixing the remaining of the $\mathbf{A}^{(d-1)}$ factor matrices until convergence to solve the following optimization function:

$$\min_{\mathbf{A}^{(d)}} ||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{M}}||^2 \tag{3.6}$$

The second tensor decomposition algorithm used in our studies is CANDECOMP/PARAFAC Alternating Poisson Regression (CP-APR), a non-negative tensor decomposition method, which minimizes Kullback-Leibler (KL) divergence via an updated Multiplicative Update (MU) algorithm [12]. The CP-APR algorithm includes a non-negativity constraint which allows the latent factors to be additive parts of the original data resulting in an improved interpretability. In CP-APR, tensor is modeled under Poisson distribution with the Poisson rate parameter $\gamma > 0$ such that:

$$\boldsymbol{\mathcal{X}}_{\mathbf{i}} \sim \text{Poisson}(\gamma_{\mathbf{i}}) \tag{3.7}$$

CP-ALS algorithm was first released in the popular tensor decomposition software named MATLAB Tensor Toolbox [23]. With RFoT, we introduce a Python implementation of CP-ALS, which is used during our experiments. For the CP-APR algorithm, we use the previously introduced Python implementation with GPU capability [26]. For more information on tensors, we refer the reader to [11, 65]. More details of the CP-APR algorithm can be found in [12], and details of CP-ALS can be found in [9].

3.2 Clustering Specimens Over the Latent Components

We find that malware and benign-ware samples can be separated in an unsupervised manner using tensor decomposition. In this section, we first describe the tensor configuration needed to obtain sample groupings from a tensor decomposition. We also show real examples of malware and benign-ware clusters in the latent factors. We then summarize the clustering methods used in RFoT to capture the patterns formed in the latent factors.

3.2.1 Malware Patterns in the Latent Factors

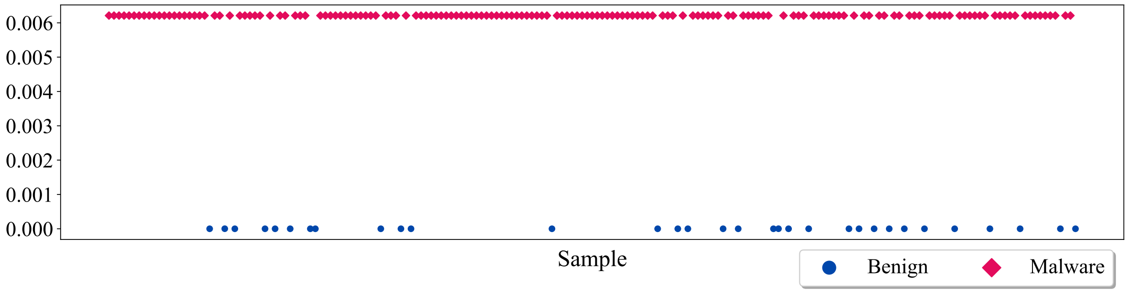

Figure 3.2: Clean malware and benign-ware clusters found by tensor decomposition

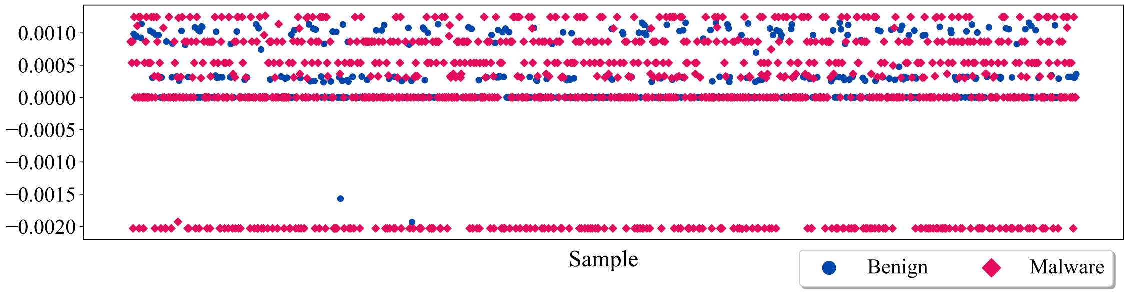

Figure 3.3: Clean malware clusters and noisy benign-ware clusters found by tensor decomposition

In order to extract latent factors with the capability of describing malware and benign-ware patterns based on each sample individually, we set our first dimension of the tensor $\boldsymbol{\mathcal{X}}$ to represent each malware sample while selecting the remaining of the $D-1$ dimensions and the tensor entry from static-malware-analysis based features using the PE header. We give a detailed description of how the remaining $D-1$ is configured to build the tensor from PE features in Section 3.3. In this tensor configuration, the shape of $\boldsymbol{\mathcal{X}}$ is $N_1 \times N_2 \times \cdots \times N_D$, where N_1 is the total number of malware and benign-ware files from our dataset. For example, to access the tensor entry of the first specimen from the dataset, for features that are indexing at $i_2, \cdots, i_D$, we would index the tensor as $\boldsymbol{\mathcal{X}}_{0,i_2,\cdots,i_D}$. Because the first dimension of $\boldsymbol{\mathcal{X}}$ represents the samples, the obtained latent factor matrix for mode-1 is $\boldsymbol{\mathcal{M}}_{::0} = \mathbf{A}^{(1)} \in \mathbb{R}^{\mathrm{N_1 \, x \, R}}$, where R is the tensor rank and $\mathbf{A}^{(1)}$ carries latent information regarding the samples in our data. Using $\mathbf{A}^{(1)}$, we can access each individual latent factor $\mathbf{A}^{(1)}_{:r} \in \mathbb{R}^{\mathrm{N_1 \, x \, 1}}$ in which the N_1 malware and benign-ware samples would form clusters. In Figure 3.2, we provide an example latent factor $\mathbf{A}^{(1)}_{:r}$ obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. Here, we can see that CP-ALS was able to cleanly separate malware and benign-ware instances within the latent factor. We also provide a second example with more noisy clusters in Figure 3.3. In Figure 3.3, around 7 lines forming clusters can be seen. Although the lines that have the majority of the samples from the malware class form cleaner clusters, there are other clusters where benign and malware samples are included within the same cluster. In Section 3.4.2, we will describe how we handle the more noisy clusters, or the clusters with poor uniformity where the majority of the cluster is not represented by a single class, using the *Cluster Uniformity Score*.

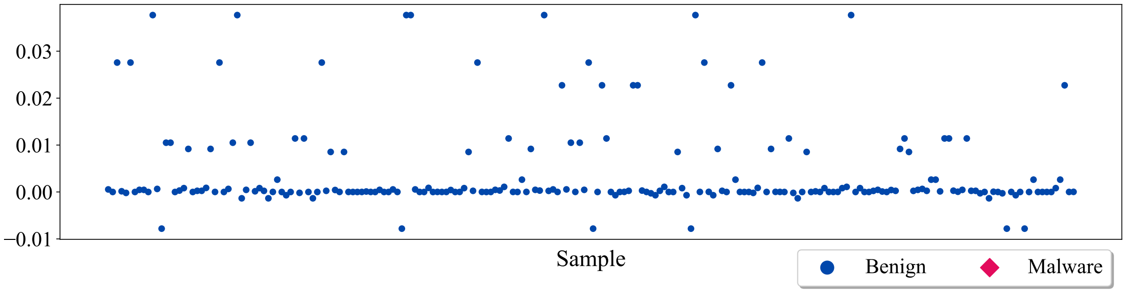

Figure 3.4: Tensor decomposition placing benign samples into a single latent factor

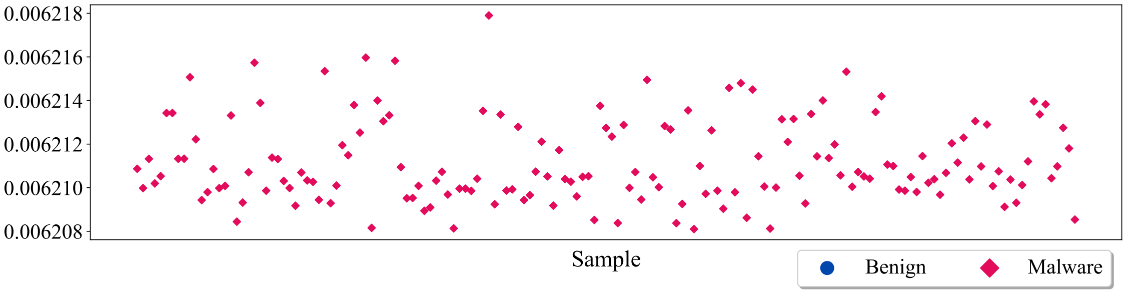

Figure 3.5: Tensor decomposition placing malware specimens into a single latent factor

In addition to observing clusters that separate malware and benign-ware within each latent factor $\mathbf{A}^{(1)}_{:r}$, we also find that malware and benign instances are cluster among components in $\mathbf{A}^{(1)}$, such that a single component r represents samples from a single class. For instance, in Figure 3.4 we again show a latent factor obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. This time, it can be seen that CP-ALS was able to cluster benign instances within a single factor from the component r. Similarly, in Figure 3.5, it can be seen that the latent factor r only contains malware specimens. Motivated by the fact that we can acquire meaningful patterns that distinguish malware and benign-ware using tensor decomposition, we next look at how these patterns can be captured to derive a functionality that enable building a semi-supervised classifier.

3.2.2 Capturing the Latent Patterns via Clustering

The performance of the RFoT depends on the success in capturing the patterns found by tensor decomposition into clusters. Therefore, in this thesis, we compare two different clustering methods. The first clustering algorithm we use to capture the patterns is called Mean Shift (MS) [13–15]. Specifically, we use the Scikit-learn implementation of this algorithm [16]. MS uses centroids to be the mean of clusters and updates the location of the clusters in a hill-climbing fashion to locate the maxima of a given density function, making it a good fit to perform clustering in a 1-dimensional space [16]. The window length, or the furthest point from the centroid of a cluster, is selected via the bandwidth. We use Scikit-learn's *estimate_bandwith*[1] API to automatically determine the number of clusters. RFoT applies MS to each component within the latent factor for the first dimension $\mathbf{A}^{(1)}_{:r}$ from a given tensor decomposition. We extract total of G_r clusters from each latent factor $\mathbf{A}^{(1)}_{:r}$, adding up to total of $G = G_0 + G_1 + \cdots + G_{R-1}$ clusters for a single tensor decomposition. We let the $\mathbf{g}_{j,r}$ represent a cluster with a set of samples from the rth component and jth cluster, where $0 \leq j \leq G_r - 1$.

In addition to the MS clustering, we introduce the *Component* clustering in this thesis. The motivation behind the Component clustering comes from our observation that we can obtain class based groupings among components, rather than within individual latent factor, obtained from tensor decompositon. We discussed this in Section 3.2, where figures 3.4 (component with only benign samples) and

---

[1] We heuristically set the *quantile* hyper-parameter to be 0.1 for estimating the bandwidth.

3.5 (component with only malware specimens) showed an example of clean clustering within a single latent factor from the component r. Formally, when using the *Component* clustering, we will let each $\mathbf{A}^{(1)}_{:r}$ to define a single cluster such that $\mathbf{g}_{r,r} = \mathbf{A}^{(1)}_{:r}$. The total number of clusters from the component r in this case is $G_r = 1$, and the total number of clusters for the decomposition is $G = R$, where R is the tensor rank. Recall that figures 3.2 and 3.3 showed example latent factors where we had mix of both malware and benign-ware clusters. We will also use the *Cluster Uniformity Score*, introduced below at Section 3.4.2, to filter out the cases where we have more than one class describing a single latent factor.

After each tensor decomposition, we apply pre-procesing to each latent factor $\mathbf{A}^{(1)}_{:r}$ to keep the samples with signals, or samples with a value that is not near 0 within the latent factor. To this end, prior to applying MS or Component clustering, we mask out (or remove) the points that are close to zero, where the distance to 0 is controlled with the hyper-paremeter *zero_tol* in RFoT. In our experiments, we set $zero\_tol = 1e-08$.

3.3 Ensemble of Random Tensor Configurations

3.3.1 Notation for an Ensemble of Tensors

Patterns extracted with tensor decomposition depends on the configuration of the tensor including the selection of the dimensions, tensor entry, and tensor rank. RFoT uses the *"wisdom of crowds"* philosophy by utilizing the patterns found from an ensemble of tensor configurations with randomly selected dimensions, entries, and ranks. We use the variable *n_estimators* to represent the number of randomly generated tensor configurations. Let $\boldsymbol{\mathcal{X}}^{(i)}$ be one of the randomly generated tensors where i is in range $1 \leq i \leq n\_estimators$. To describe the random tensor configuration members of an ensemble, we re-formulate the notations introduced for tensor decomposition in Section 3.1. We begin with re-writing the CPD formula with sum of rank-1 tensors:

$$\boldsymbol{\mathcal{X}}^{(i)} \approx \sum_{r=1}^{R_i} \lambda_r \cdot \mathbf{a}_r^{(i,1)} \circ \mathbf{a}_r^{(i,2)} \circ \cdots \circ \mathbf{a}_r^{(i,D_i)} \tag{3.8}$$

Here we have the rank R_i CPD for the ith random tensor $\boldsymbol{\mathcal{X}}^{(i)}$ with D_i dimensions, and each $\mathbf{a}_r^{(i,d)}$ represents the rth latent factor for dimension d, where r is in range $1 \leq r \leq R_i$ and d is in range of $1 \leq d \leq D_i$. Following the KRUSKAL format we re-write the low-rank approximation as follows:

$$\boldsymbol{\mathcal{X}}^{(i)} \approx \boldsymbol{\mathcal{M}}^{(i)} \equiv [\![\lambda\, ;\ \mathbf{A}^{(i,1)}, \mathbf{A}^{(i,2)}, \cdots, \mathbf{A}^{(i,D_i)}]\!] \tag{3.9}$$

Here $\boldsymbol{\mathcal{M}}^{(i)}$ is the low-rank estimation for the ith random tensor, and $\boldsymbol{\mathcal{M}}^{(i)}_{::d-1} = \mathbf{A}^{(i,d)}$ is the latent factors matrix for dimension d. Each $\mathbf{A}^{(i,d)} \in \mathbb{R}^{\mathrm{N_d \, x \, R_i}}$ is a collection of latent factors as follows:

$$\mathbf{A}^{(i,d)} = [\mathbf{a}_1^{(i,d)}, \mathbf{a}_2^{(i,d)}, \ldots, \mathbf{a}_{R_i}^{(i,d)}] \tag{3.10}$$

As explained in Section 3.2.1, to capture the sample groupings, RFoT fixes the first dimension to represent each sample from our dataset. In an ensemble of random tensor configurations setting, $\mathcal{M}^{(i)}_{::0} = \mathbf{A}^{(i,1)} \in \mathrm{I\!R}^{\mathrm{N_1 \, x \, R_i}}$ is the latent factors matrix for the first dimension representing the N_1 malware and benign instances for the ith random tensor. MS clustering is applied to each $\mathbf{A}^{(i,1)}_{:r} = \mathbf{a}^{(i,1)}_r$, to capture $G^{(i)}_r$ number of clusters from component r, such that the total number of clusters found from ith tensor is $G^{(i)} = G^{(i)}_0 + G^{(i)}_1 + \cdots + G^{(i)}_{R_i-1}$. In an ensemble notation, we will let each cluster with a set of samples to be denoted with $\mathbf{g}^{(i)}_{j,r}$, where $0 \leq j \leq G^{(i)}_r$, for the rth component of ith tensor decomposition. With Component clustering, each cluster is $\mathbf{g}^{(i)}_{r,r} = \mathbf{A}^{(i,1)}_{:r}$, and the total number of clusters $G^{(i)} = R_i$ for the ith tensor decomposed to rank R_i.

3.3.2 Random Tensor Configuration Sampling

Our random tensor sampling includes a random selection of the number of dimensions, the features to represent each dimension, tensor entry, and random or fixed tensor rank. For each i random tensor, we first randomly choose the number of dimensions D_i with replacement, such that the range of D_i is $3 \leq D_i \leq \beta - 1$, where β is the total number of features from the original matrix $\mathbf{X} \in \mathrm{I\!R}^{\mathrm{N_1 \, x \, \beta}}$. The minimum and maximum number of dimensions a random tensor configuration can have is controlled using the RFoT hyper-parameters (*min_dimensions*, *max_dimensions*). Here *min_dimensions* >= 3 since tensors have at least 3 dimensions, and *max_dimensions* <= β since we need one of the features to be the tensor entry. The first dimension is size N_1 representing each malware and benignware sample, and the features representing the remaining $D_i - 1$ dimensions are randomly selected from β features without replacement. Next, from the remaining $\beta - D_i$ features, which represent the feature(s) that are not selected to be a tensor dimension, RFoT randomly selects the feature to be used as the tensor entry with replacement. Finally, rank R_i is selected randomly, also with replacement, or each random tensor is assigned a user-defined fixed rank $R_i = rank$.

Tensor rank determines the number of hidden features, or latent components, that the tensor decomposition should extract. If we choose the rank to be too low, then we miss vital information (*under-fitting*), while if the rank is chosen to be too high, then we include noise in our solution (*over-fitting*) [66]. If we under-fit or over-fit the solution, then the latent factors might not have meaningful patterns to cluster benign and malware instances. By randomly selecting the rank, we attempt to avoid the need to correctly determine the rank of the tensor. Using the philosophy *"wisdom of crowds"*, we hope that an ensemble of tensor decomposition collectively can reach to a consensus in the extracted patterns and allow precise malware detection. By doing so, we let the ensemble members, random tensors, to complement each others' weaknesses. Specifically, we want to cancel out the impact of the cases where we obtain poor tensor decomposition results by deriving a decision based on the majority of the population. This hypothesis assumes that the cases with impure clusters does not represent the majority of the population.

As we sample each random tensor configuration, we do not check if the same configuration was already used. This check is avoided to ensure that the random sampling process remains fast as the size of the ensemble grows. Therefore, the aforementioned 4 steps for sampling random tensor configurations for building the ensemble of $n\_estimators$ tensors can result in repeated tensor configurations, which would need to be discarded after the sampling. Inspired from the previously introduced technique for fast sampling of zero tensor indices [63], we over-sample the tensor configurations to lower the probability of the repeated tensor configurations. Therefore, we set the ensemble size to be $n\_estimators + (n\_estimators \cdot 0.1)$ random tensors. We then perform post-processing to keep the unique tensor configurations and under-sample the ensemble to be the size of at most $n\_estimators$.

3.3.3 Feature to Tensor Dimension Mapping and Tensor Entry

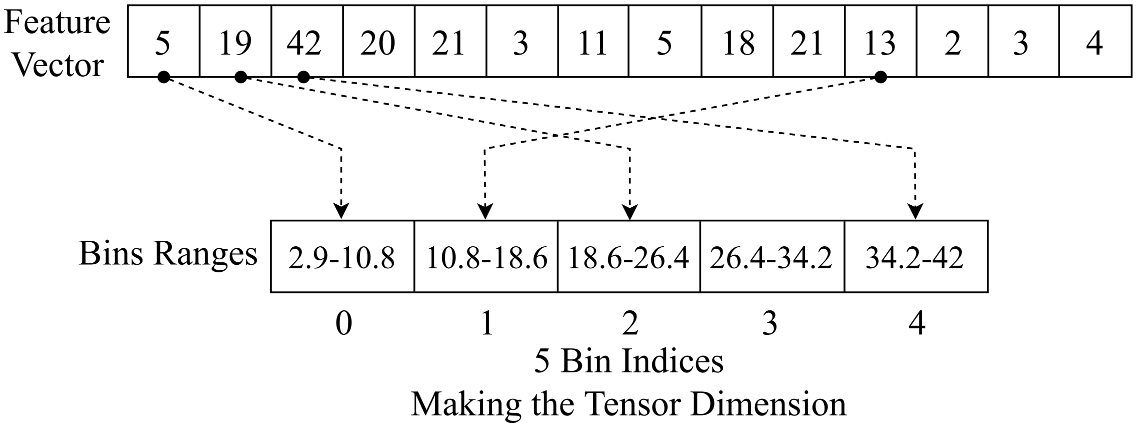

Figure 3.6: Example of 4 numerical feature values being mapped to 5 bins to form a tensor dimension.

Categorical features can easily be mapped to an index in the tensor dimension. For example, take the EMBER-2018 feature *has_signature*, a binary feature that can be a 0 or 1. If a tensor dimension represents this feature, the size of that dimension would be 2, where $has\_signature = 0$ would map to index 0, and $has\_signature = 1$ would map to index 1. This generalizes to any categorical feature where the labels can be encoded to retrieve features to dimension index mapping.

On the other hand, in order to use a numerical feature as a tensor dimension we need to utilize binning to map the given numerical value to a certain index in the dimension. RFoT uses *cut* API of *Pandas* Python library to bin numerical values [67,68]. The number of bins, or dimension size N_d, is determined by the RFoT hyper-parameter *bin_scale*, where the number of bins is $N_d = \mathit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$. Here $\text{num\_unique}(\mathbf{f})$ gives the total number of unique elements present in a given feature vector $\mathbf{f}$, which is one of β features. We provide an example in Figure 3.6 which shows how numerical features are mapped to 5 bins to from a tensor dimension.

We can utilize a real example to further explain how a tensor for malware data can be build using numerical feature binning and categorical feature mapping. For example, let ith random tensor $\mathcal{X}^{(i)}$ have the dimensions *Sample - Number of Strings - Has Signature*, and entry *Number of Sections. Sample* dimension represents each N_1 malware and benign-ware sample. The dimension *Number of Strings* represents the number of printable strings present in a given malware or benign instance, with size $N_2 = \mathit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$, such that the EMBER-2018 features for the number of strings will map to an index between 0 and $N_2 - 1$. The categorical dimension *Has Signature* identifies if a given specimen has a signature or not, thus the size of the last dimension is $N_3 = 2$. Finally, the tensor entry *Number of Sections* determines the number of sections present in the PE header of a given file. An entry $\mathcal{X}^{(i)}_{n,s,f}$ in this tensor represents the number sections that a specimen $n \in [0, 1, \cdots, N_1 - 1]$, with number of strings that bins to an index $s \in [0, 1, \cdots, N_2 - 1]$, and with the signature flag $f \in [0, 1]$ has.

3.4 Semi-supervised Classification with RFoT

Tensor decomposition extracts latent patterns from multi-dimensional data in an unsupervised fashion, and we capture these patterns for malware and benign samples using clustering techniques as described in Section 3.2. Using the captured clusters, we formulate a semi-supervised classifier that utilizes the information found by tensor decomposition. In this section, we first describe how the semi-supervised voting over the clusters is performed. This include the cases where the model is unable to make a decision for a given sample, and thus *abstaining vote* is given. We then introduce the *Cluster Uniformity Score* that is used as a threshold to filter out noisy, or non-uniform clusters.

3.4.1 Semi-supervised Voting Using the Clusters

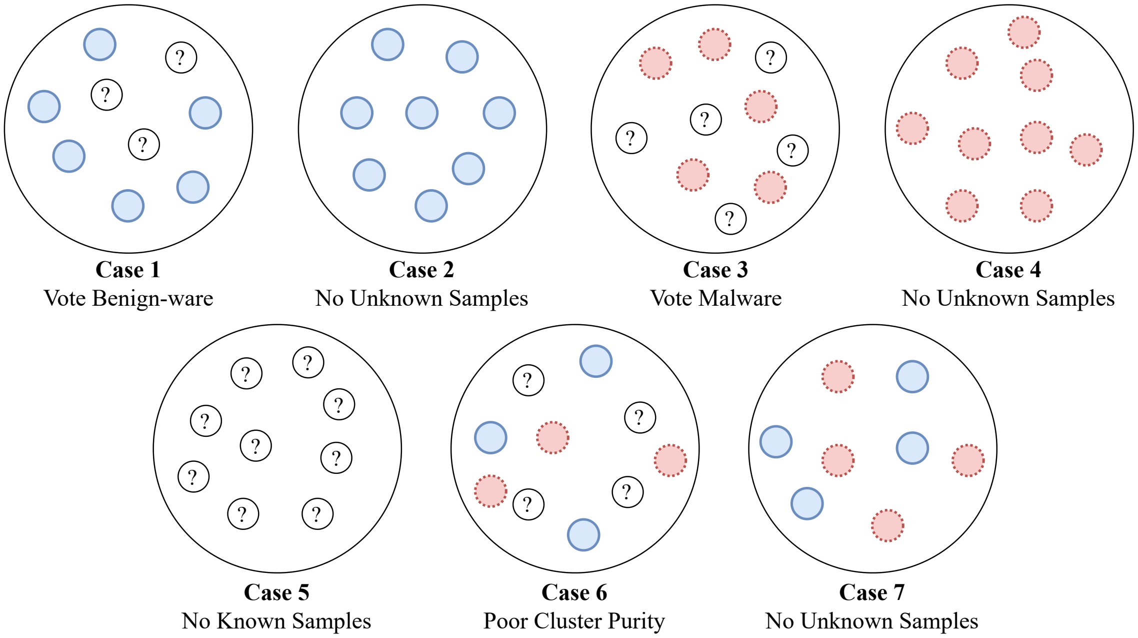

Figure 3.7: Possible cases of clusters that can be seen

RFoT takes a dataset $\mathbf{X} \in \mathrm{I\!R}^{\mathrm{n\,x}\,\beta}$, where n is the number of samples and β is the number of features, and a vector $\mathbf{y}$ that represents the labels for each n samples such that $y_n \in [-1, 0, 1, \cdots, C-1]$. Note that -1 is used for the unknown specimens, and C is the number of classes. In this thesis, we have $C = 2$ for malware and benign-ware, such that $y_n \in [-1, 0, 1]$ where 0 labels the benign-ware and 1 labels the malware. When we obtain a cluster, we use the known samples (samples with labels) as a reference to help us make a decision against the unknown samples (samples without the labels, or -1) within that cluster. Specifically, the class vote of the given unknown samples that are in the cluster $\mathbf{g}_{j,r}^{(i)}$ is determined by the majority class of the known samples that are in the same cluster $\mathbf{g}_{j,r}^{(i)}$. There are 7 possible cases of cluster characteristics that we can obtain from the latent components, which are shown in Figure 3.7. In **Case 1**, we may have a cluster containing a set of unknown specimens and a set of known benign-ware. In this case, we would vote the unknown specimens as benign file. Similarly, we can vote the unknown specimens as malware if the majority of the known specimens are malware in the same cluster, as shown in **Case 3**. It is also possible to come across with clusters where no unknown specimens are present, as shown in **Case 2**, **Case 4**, and **Case 7**. If there are no unknown samples in a given cluster, we disregard the cluster since we do not need to perform any voting.

This semi-supervised setup for classifying unknown specimens via clustering allows us to perform abstaining predictions (i.e. predict *"I do not know"*) due to not being able to obtain a class vote for a given sample. For instance, if a cluster consisting of only a set of unknown specimens, as shown in **Case 5**, we cannot take a vote for these samples since we do now have any labeled instances to inform us regarding the class vote. We also cannot take a vote for the samples that are masked out due to the lack of signals (samples that are close to 0 with a certain threshold), as described in Section 3.2.2, since these instances would not be used in clustering. If a given sample always falls in a cluster without any known samples, like in **Case 5**, for each random tensor $\boldsymbol{\mathcal{X}}^{(i)}$ and its latent factor for the first dimension $\mathbf{A}^{(i,1)}$ obtained by the decomposition, then this sample is predicted to be abstaining, or its label is kept as unknown (-1). Similarly, if a given sample n is consistently masked out due to being near zero in each $\mathbf{A}^{(i,1)}$, then it is predicted to be abstaining.

The samples that do get class vote(s), we perform max-vote to determine the final class prediction. That is, if a given specimen n has its majority of the votes (over 50%) representing one of the C classes, the instance n is predicted to be that class.

3.4.2 Cluster Uniformity Score

It is possible to encounter a cluster that is not uniform in representing a single class (cluster have known instances from multiple classes). We have already shown an example of a latent factor with non-uniform clusters in Figure 3.3, where noisy clusters occur. In Figure 3.7, **Case 6** demonstrates a cluster where we have a mix of known malware and benign-ware specimens. In such cases, we cannot obtain an accurate class vote from the cluster. To filter out these clusters, we use the *Cluster Uniformity Score* which is calculated based on the fraction of the most dominant known class in the given cluster $\mathbf{g}_{j,r}^{(i)}$. We have previously introduced cluster uniformity score in our prior work, that is under review at ACM TOPS journal, for determining the uniformity of the clusters based on known specimens [69]. In this thesis, we utilize the same metric, but re-formulate it to match with ensemble of tensors notation as follows:

$$U^{\mathbf{g}_{j,r}^{(i)}} = \frac{|\max\left({\mathbf{g}_{j,r}^{(i)}}^{known}\right)|}{|{\mathbf{g}_{j,r}^{(i)}}^{known}|} \tag{3.11}$$

Here $U^{\mathbf{g}_{j,r}^{(i)}}$ is the cluster uniformity score for jth cluster obtained from rth component of tensor decomposition of ith tensor, $\mathbf{g}_{j,r}^{(i)}$. $|\max({\mathbf{g}_{j,r}^{(i)}}^{known})|$ is the number of samples that belongs to the most dominant class with known samples in the cluster $\mathbf{g}_{j,r}^{(i)}$, while $|{\mathbf{g}_{j,r}^{(i)}}^{known}|$ is the total number of known samples in the cluster. The clusters where $U^{\mathbf{g}_{j,r}^{(i)}}$ is below the specified uniformity threshold t are removed from consideration, and thus no class vote is obtained from these clusters. If a given specimen n continuously falls in the clusters that are removed due to poor purity, it is also predicted to be abstaining at the end.

3.5 Putting it Together: RFoT Algorithm

We summarize the RFoT methodology in Figure 3.8, and with a pseudo-code in Algorithm 1. We first randomly sample tensor configurations (1). Then each tensor configuration is factorized to obtain the latent components (2). Within each latent component, we look at the latent factor representing each malware and benign-ware sample (2). Clustering is applied to capture the groupings within each of these latent

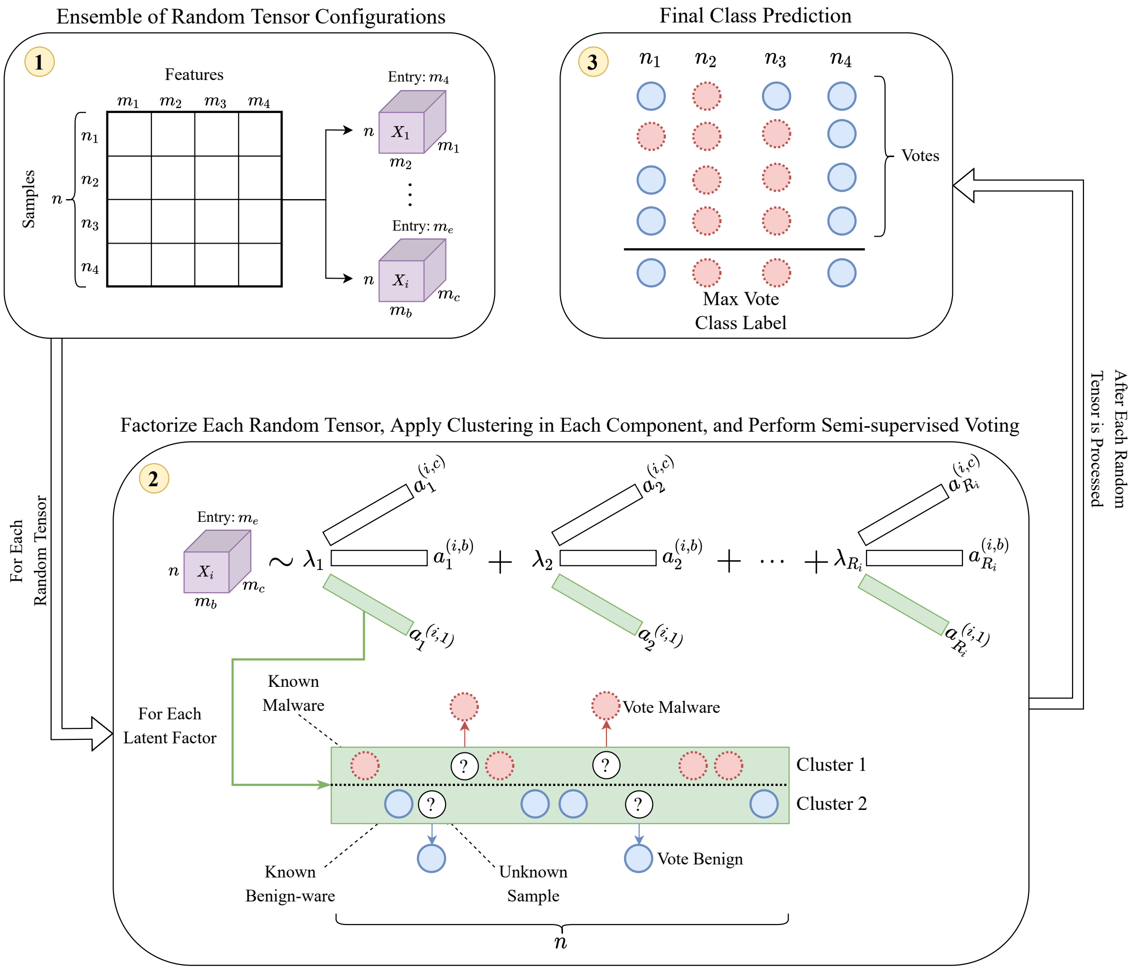

Figure 3.8: RFoT methodology overview

factors (2). We filter out the noisy clusters using the cluster uniformity score. In the cases where we were able to acquire clean clusters, we take a class vote in a semi-supervised fashion (2). After each tensor is factorized, and class votes are obtained from each latent factor for the first dimension, we get the final class prediction via max-vote (3). The specimens are predicted to be abstaining if they did not get any class vote due either being part of clusters that were not uniform, not falling in a cluster that had known samples, or because they were masked out due to not having a signal.

Algorithm 1 RFoT($\mathbf{X}$, $\mathbf{y}$, *n_estimators*, *bin_scale*, t, R, *min_dims*, *max_dims*)

```
 1: tensor_configs = sample_tensors($\mathbf{X}$, $n\_estimators$, $R$, $min\_dims$, $max\_dims$)
 2: class_votes = []
 3: for config in tensor_configs do                              ▷ Start the parallel execution
 4:     $\mathcal{X}^{(i)}$ = build_tensor(bin_features($\mathbf{X}$, config), config)      ▷ COO format
 5:     $\mathcal{M}^{(i)}$ = decompose($\mathcal{X}^{(i)}$, $R_i$)                         ▷ CP-ALS or CP-APR
 6:     $\mathbf{A}^{(i,1)}$ = get_signals($\mathcal{M}^{(i)}_{::0}$)      ▷ Mask out near zero elements for the mode-1
 7:     clusters = cluster_latent_factor($\mathbf{A}^{(i,1)}_{:r}$)      ▷ For each $R_i$, MS or Component
 8:     for $\mathbf{g}^{(i)}_{j,r}$ in clusters do
 9:         if $\mathbf{g}^{(i)}_{j,r}$ in [Case 4,5,6 or 7] then             ▷ See Figure 3.7 for the cases
10:             continue                                              ▷ Abstaining votes
11:         else
12:             class_votes.append(vote($\mathbf{g}^{(i)}_{j,r}$, $\mathbf{y}$))          ▷ Semi-supervised voting
13:         end if
14:     end for
15: end for                                                      ▷ End the parallel execution
        $\mathbf{y}_{\text{pred}}$ = max_vote(class_votes, $\mathbf{y}$)                   ▷ Final class prediction
16: return $\mathbf{y}_{\text{pred}}$
```

We finally note that our implementation of RFoT computes the decomposition of the ensemble of random tensor configurations in a parallel fashion, since they are independent of one another. The parallel computation of the members of ensemble allows us to reduce the total time needed for prediction. Specifically, in Algorithm 1, lines 3 through 15 are executed in parallel based on the number of jobs that the user wants to run.

Now that we have introduced our methodology, we will next showcase the experiment results from a case-study where we classified malware and benign samples from the EMBER-2018 dataset using RFoT.

Chapter 4: Experiments

In this section, we first introduce the dataset and the features used in our experiments. Next, we summarize the performance evaluation metrics used in our studies. Our experiments include the evaluation of RFoT performance under different hyper-parameter settings. We then compare our results to the baseline models, and test the performance of RFoT and the baseline models with the dropping percentage of the labeled data.

4.1 EMBER-2018 Dataset and Experiment Setup

For long time publicly available malware datasets for benchmarking ML methods have been limited. This is attributed to the challenges associated with obtaining labeled malware data including copyright issues for benign data, and the long and costly process of labeling malware data [8]. Anderson et al. addressed this problem and introduced the EMBER-2018 dataset [17]. Since the release of EMBER-2018, it has become a popular dataset used by researchers to evaluate their ML-based techniques within the malware field. Consequently, we use EMBER-2018 in our study to evaluate the capabilities of our introduced algorithm, RFoT.

EMBER-2018 consist of PE header and file meta-data features drawn from

1.1 million Windows malware and benign-ware, out of which 800,000 of them has labels. In this study we use 9 PE header features to construct our tensors, and train the baseline models. Specifically, following features are used:

1. *Number of Strings:* the number of printable strings in the file
2. *Strings Entropy:* randomness measurement for the printable strings
3. *File Size:* size of the executable in bytes
4. *Number of Exports:* number of functions being exported by the binary
5. *Number of Imports:* number of functions being imported by the binary
6. *Size of Code:* the size of the code section (.txt) in PE header
7. *Number of Sections:* the total number of sections in the PE header
8. *Has Debug:* flag to indicate if the debug value is on/off
9. *Has Signature:* flag to indicate if the file has a signature

To conduct our experiments, we build 10 smaller subsets out of all the 800,000 instances in EMBER-2018, where each subset contains 10,000 balanced amounts of malware and benign-ware. We apply our experiments to every 10 subset data to show that the reported results are statistically significant. Our final results are reported with a 95% Confidence Interval (CI). In Section 4.4, where the several evaluation metrics used to report the performance of our method as compared to the baseline models, we make test set size to be a 30% of 10,000 malware and benign instances for each 10 subsets. For RFoT, 30% test size gives us the number of samples without labels (unknown set).

4.2 Performance Evaluation Metrics

4.2.1 Precision, Recall, and F1 Measure

To evaluate the performance of our method and the baseline models, we use Precision, Recall, and F1 score. Precision score measures the ability of the model's correctly identify the positive class and can be calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1}$$

where TP is the number of true positive predictions, FP is the false positives. The Recall metric measures the extend to which model can detect the positive class, malware in our case, and it is calculated as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.2}$$

where FN is the number of false negatives. F1 score is calculated using both the Precision and Recall scores together; therefore, F1 score is only high when both Precision and Recall are high. F1 score can be calculated as follows:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

More specifically, F1 can be calculated as follows:

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.4}$$

4.2.2 Execution Time and Abstaining Predictions

The other two metrics used in our results are the execution time for both RFoT and baseline models, and the percent of abstaining predictions for RFoT. The execution time for the baseline models *XGBoost*, *LightGBM*, and *XGBoost+SelfTrain* (the semi-supervised extension of *XGBoost*) include the time it took to train the models and make predictions. Finally, the percentage of the abstaining predictions gives tells us the percent of unknown samples that remained unknown (or with label -1) after the prediction.

4.3 RFoT Hyper-parameter Performance Analysis

Prior to the comparison of our approach to the baseline models, we first evaluate its behaviour under different hyper-parameters including *Bin Scale*, *Cluster Uniformity Threshold*, *Max and Min Number of Dimensions*, *Tensor Rank*, and *Number of Estimators*. The analysis into the hyper-parameter settings allow us to learn the best model setting for the dataset that we use, and also to understand the limitations and capabilities of RFoT. We conduct the hyper-parameter analysis using the CP-ALS tensor decomposition algorithm, and MS and Component clustering methods. For CP-ALS, we set the maximum number iteration to be 250 for each experiment.

4.3.1 Bin Scale

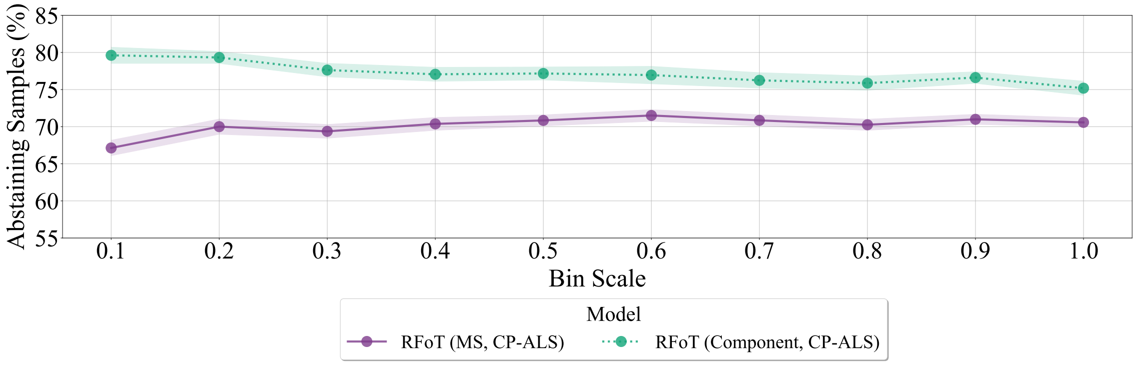

Figure 4.1: Abstaining percentage is shown for different values of the Bin Scale.

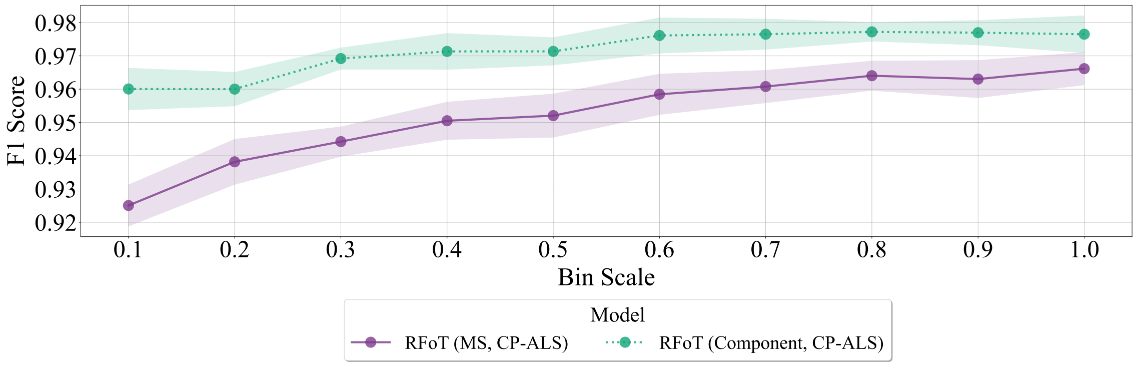

Figure 4.2: F1 score is shown for different values of the Bin Scale.

The first hyper-parameter that we investigate is *Bin Scale*, which determines the size of the tensor dimensions that represent the features with numerical values, as described in Section 3.3.3. During this analysis, we perform prediction while changing the bin_scale hyper-parameter between 0.1 and 1.0 with the step-size of 0.1. We set the remaining of the hyper-parameters as follows: $min\_dimensions = 7$, $max\_dimensions = 8$, $cluster\_purity\_tol = 1.0$, $rank = 2$, and $n\_estimators = 1000$.

Here *min_dimensions* and *max_dimensions* determines the minimum and the maximum number of dimensions any random tensor can have respectively. Since this experiment evaluates the bin scale hyper-parameter which only effects the features with numerical values, to ensure that each random tensor would have a dimension representing a feature with a numerical value we set the minimum number of dimensions to be 7. *cluster_uniformity_tol* gives us the threshold to select any given cluster to be used in voting, as described in Section 3.4.2. Using the hyper-parameter *rank*, we set each random tensor to be decomposed with rank-2. Finally, *n_estimators* determines the number of random tensors to be used.

In Figure 4.1, we can see that as the bin scale is increased, the percent of abstaining predictions for RFoT with Component clustering drops from around 80% to 75% while it increases from around 66% to 70% for MS clustering. At the same time, Figure 4.2 shows that the F1 score for RFoT with both MS and Component clustering increases as the bin scale reaches 1.0. When the bin scale is set to 1.0, the size of the dimension representing the given feature is equal to the number of unique values present in the given feature vector $\mathbf{X}_{:\mathbf{f}}$. The results in this analysis suggest that reducing the dimension size (or the number of bins) for the given numerical features, could result in under-fitting, or missing the details that help separate the malware and benign-ware. In addition, the fact that RFoT with Component clustering saw both reductions on abstaining prediction while increasing F1 score further supports RFoT being able to make a more precise decision for the given labels when we use a higher bin scale value.

Notice that RFoT with Component clustering yields better malware-detection results, but higher abstaining predictions. This could be attributed to MS clustering being able to extract meaningful clustering results that separate malware and malware from a single component, while the Component clustering misses the cases such as the one shown in Figure 3.2. Finally, we do see a larger performance improvement for the MS clustering, which indicates that the added information with the increased dimension size could be resulting in cleaner in-component clusters that separate classes.

4.3.2 Cluster Uniformity Threshold

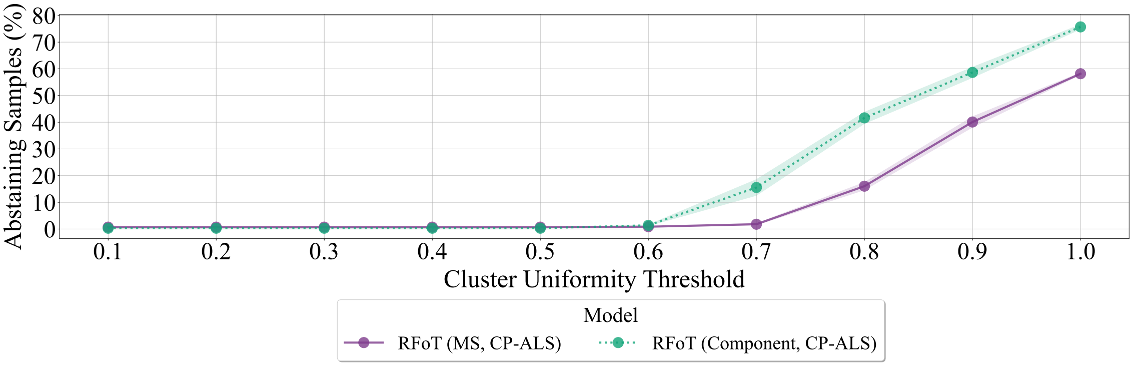

Figure 4.3: Abstaining percentage is shown for different values of the Cluster Uniformity Threshold.

We next look at the cluster uniformity threshold, which determines the threshold to remove the noisy clusters. In this analysis, the uniformity threshold is varied between 0.1 and 1.0 with a step-size of 0.1. The remaining of the hyper-parameters are kept same as the Bin Scale experiment form Section 4.3.1 except the following: $min\_dimensions = 3$ and $bin\_scale = 1$.

The first point to note in the results is that we do not get any abstaining predictions until after the uniformity threshold of 0.6 as shown in Figure 4.3. This

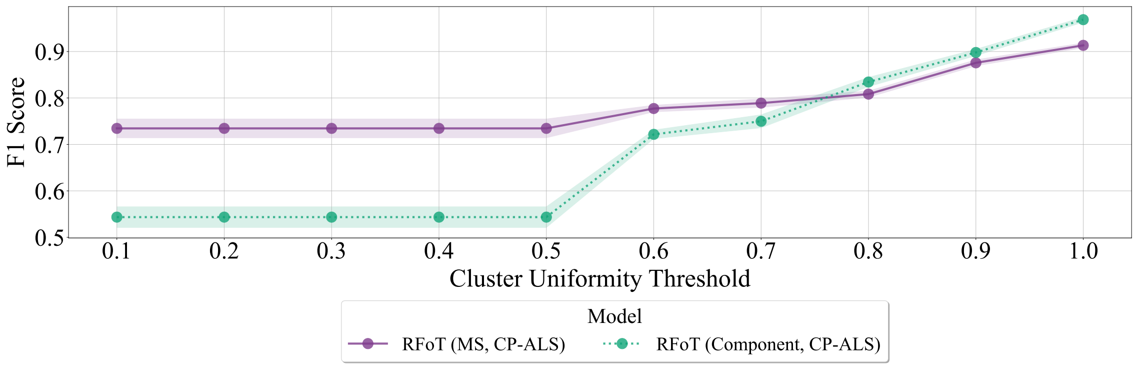

Figure 4.4: F1 score is shown for different values of the Cluster Uniformity Threshold.

results in poor performance for both MS and Component-based RFoT as shown in Figure 4.4 with low F1 scores. It can be seen, however, the F1 score improves rapidly as the cluster uniformity score is increased. The increase in the threshold also increases the abstaining predictions. This occurs because when we choose a higher cluster purity threshold, the clusters to be used in voting need to be cleaner such that the known classes in the cluster should represent mainly a single class. Specifically, it there should only be a single known class in the cluster when $cluster\_uniformity\_tol = 1$. When we encounter clusters with known samples from a mix of different classes, using the threshold, the clusters are removed from the consideration for voting. This describes the reason behind the increased abstaining predictions. Since we begin to use only the cleaner clusters, we do see the increased performance of the model.

The final point to note in this experiment is that MS-based clustering does outperform Component-based clustering with lower values of cluster uniformity threshold. This could indicate that the CP-ALS algorithm is more capable of finding meaningful in-component clusters separating classes rather than among-component clusters where each component individually separates classes.

4.3.3 Maximum and Minimum Number of Dimensions

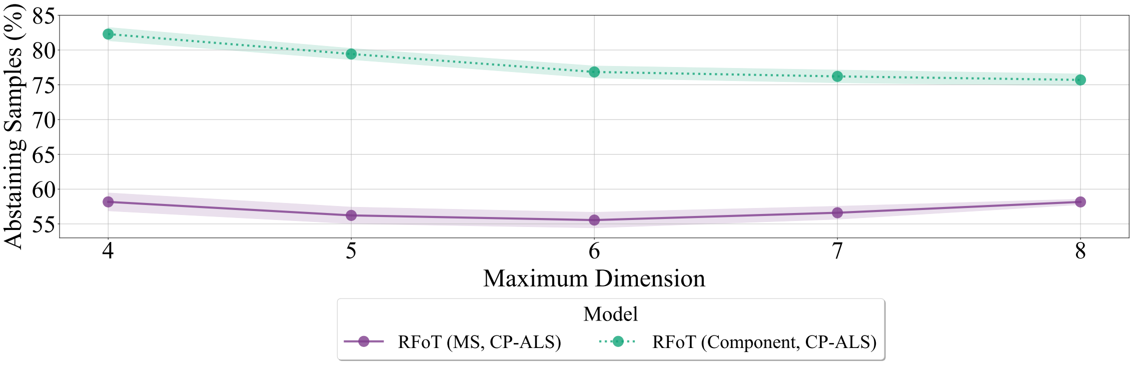

Figure 4.5: Abstaining percentage is shown for different values of the Max Dimensions parameter.

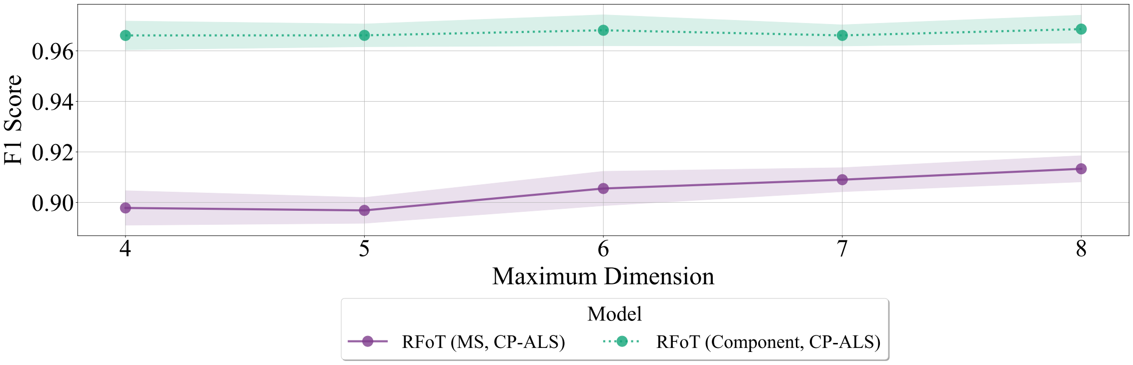

Figure 4.6: F1 score is shown for different values of the Max Dimensions parameter.

The next hyper-parameters we investigate are the choice of the minimum and the maximum number of dimensions the random tensors should have within the ensemble. To test this, we set the maximum number of dimensions between 4 and 8, with a step size of 1. For the minimum number of dimensions, we look at between 3 and 7 with a step size of 1. The remaining hyper-parameters are kept the same as the previous experiment in Section 4.3.1. Figures 4.5 and 4.6 show that we get relatively stable results for increasing maximum number of dimensions. We observe a similar trend for the Component clustering-based RFoT as the minimum

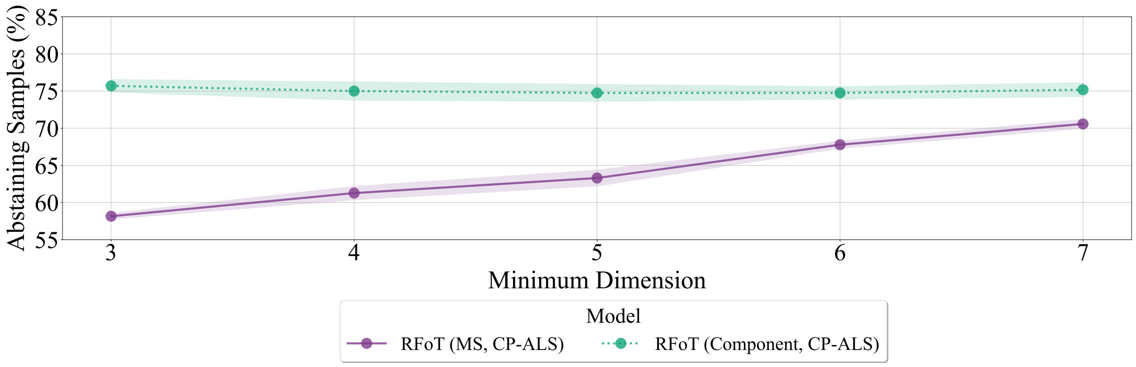

Figure 4.7: Abstaining percentage is shown for different values of the Min Dimensions parameter.

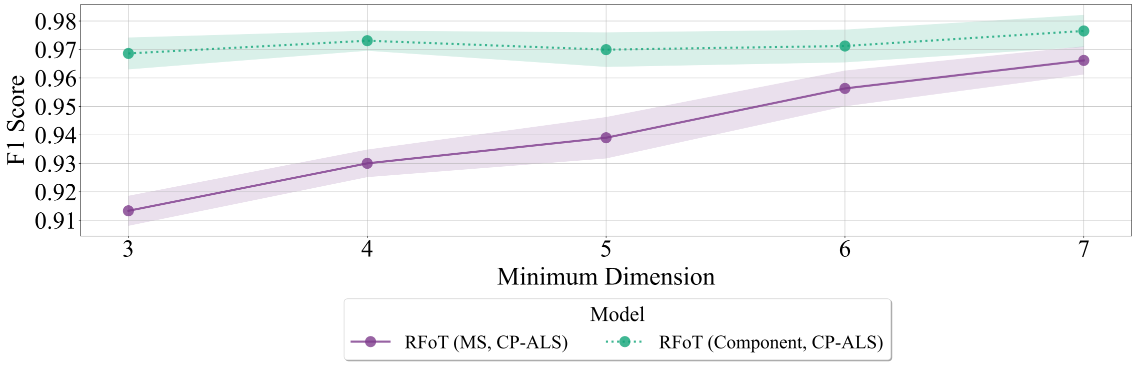

Figure 4.8: F1 score is shown for different values of the Min Dimensions parameter.

number of dimensions hyper-parameter is increased, as shown in Figures 4.7 and 4.8. However, we do see an improvement in malware classification for the MS clustering-based RFoT as the minimum number of possible dimensions is increased as shown in Figure 4.8. At the same time, there is an increase in the abstaining predictions for RFoT with MS clustering as shown in Figure 4.7. This result could indicate that CP-ALS begins to extract increased number of latent factors with noisy patterns; however, the patterns that do give meaningful result are more uniform.

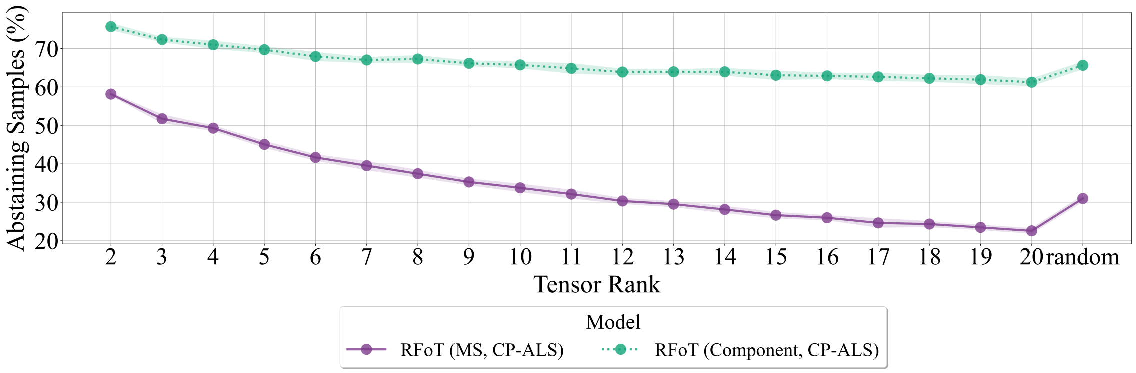

Figure 4.9: Abstaining percentage is shown for different values of the Rank parameter.

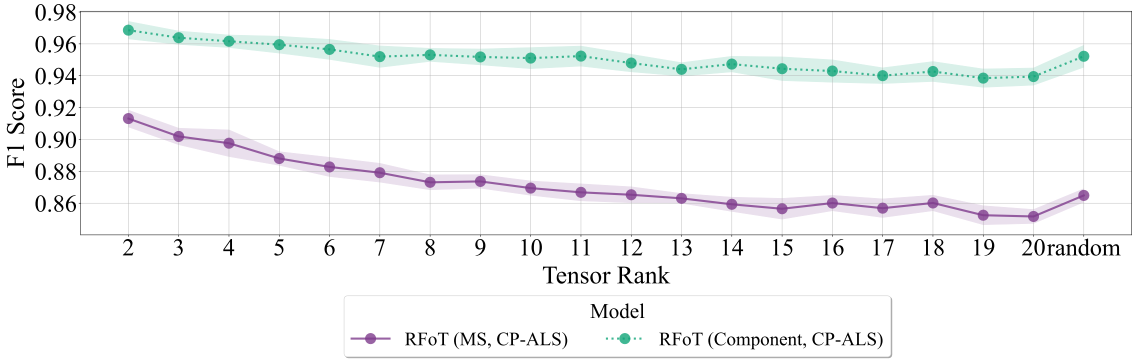

Figure 4.10: F1 score is shown for different values of the Rank parameter.

4.3.4 Tensor Rank

In our random ensemble of tensor configurations model, one of the tensor settings that can be randomly sampled is the tensor rank. We next look at the performance of RFoT with increasing fixed rank (where each tensor in the ensemble is decomposed with the same rank), and also with the randomly selected rank. The goal of this analysis to determine if the patterns extracted from an ensemble of tensor decompositions with randomly selected ranks can reach to a consensus in determining the class of the given samples, while avoiding the need to correctly determining the rank. This hypothesis also aligns with the need for the cluster uniformity calculations, where if the extracted patterns result in noisy clusters due to over-fitting or under-fitting, then we would want to remove these clusters using the cluster uniformity threshold.

For the fixed ranks, we test RFoT for MS and Component clustering where the ranks are ranged between 2 and 20 with the step size of 1. We also randomly choose the rank in this experiment for comparison to the fixed rank. The remaining of the hyper-parameters are as follows: $min\_dimensions = 3$, $max\_dimensions = 8$, $bin\_scale = 1.0$, $cluster\_purity\_tol = 1.0$, and $n\_estimators = 1000$.

In Figure 4.9, we can see that as the rank is increased the percent of abstaining predictions for both RFoT with Component and MS clustering drops. This drop is more significant for the MS clustering based RFoT. As the number of components, or the rank, is increased, RFoT obtains more opportunities to find clusters to retrieve class votes. Increase in the total number of clusters, which results in increased number of possible class votes for the unknown specimens could describe the reason behind the drop in the number of abstaining predictions. At the same time, the steeper decline in the number of abstaining predictions for RFoT with MS clustering indicates that the increasing rank improves the capability of CP-ALS to extract patterns where in-component groupings separate malware and benign-ware.

As the abstaining predictions drop with the increasing rank, the performance of the model slightly drops as shown in the Figure 4.10. We again see a higher drop, compared to the Component clustering, for the MS clustering which could be the result in the steep decline in the number of abstaining predictions. Finally, we see that random selection of rank plays a role to smooth out the results for both the F1 score and the number of abstaining predictions.

4.3.5 Number of Estimators (Random Tensors)

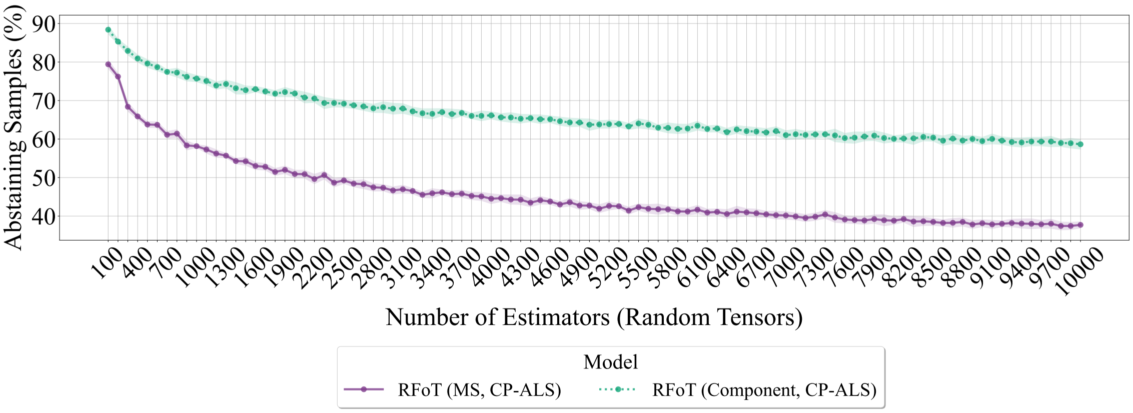

Figure 4.11: Abstaining percentage is shown for different values of the Num Estimators parameter.

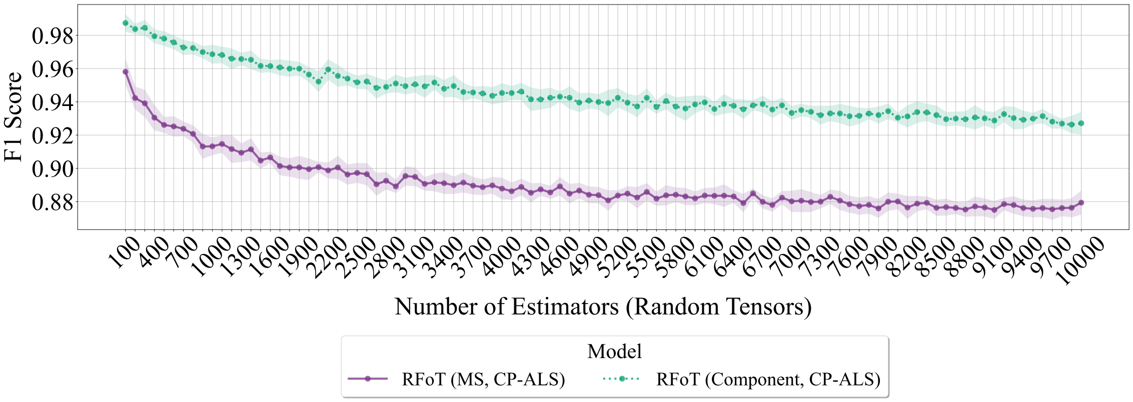

Figure 4.12: F1 score is shown for different values of the Num Estimators parameter.

The final hyper-parameter that we investigate is the selection of the total number of estimators, or number of random tensor configurations in our ensemble. We test number of estimators between 100 and 10,000 with the step size of 100. The remaining of the hyper-parameters are kept as same as the ones we used in Section 4.3.4, except we use a fixed rank of $rank = 2$. We select a low-rank of 2 motivated from the results presented in Section 4.3.4. Although we might under-fit the data, cluster purity score allows us to separate out the noisy results and only keep the meaningful patterns, which enables achieving a better malware prediction accuracy.

We observe that as the number of estimators increases, the number of abstaining predictions drops as shown in Figure 4.11. The increased number of estimators also increases the number of clusters obtained from each tensor decomposition where we can potentially obtain class votes. The dropping number of abstaining predictions with the increasing number of estimators could be due to the increase in the votes. With the dropping number of abstaining predictions, we also see a decline in F1 score as shown in Figure 4.12. However, for both the F1 score and the number of abstaining predictions, we see that the decline begins to flatten. Therefore, although the increased number of votes results in a drop in performance, as we begin to obtain more votes from a larger ensemble, the model begins to make better decisions and slow down the performance decline. The performance of our model converging to a plateau shows that RFoT is capable of using the decision made from a majority of random tensors to obtain accurate predictions when the size of our ensemble is large enough. Differently, when the ensemble is small, RFoT is more certain (accurate) for the predictions made, but a smaller number of samples are predicted due to the high abstaining prediction percentage.

Table 4.1: Baseline comparisons

Model	**Method**	**F1**	**Precision**	**Recall**	**Abstaining (%)**	**Time (sec)**
RFoT (Component, CP-ALS)	Semi-supervised	**0.968** (+-0.005)	**0.968** (+-0.005)	**0.968** (+-0.006)	75.703 (+- 0.863)	536.151 (+- 5.132)
RFoT (MS, CP-ALS)	Semi-supervised	0.913 (+-0.005)	0.915 (+-0.004)	0.913 (+-0.005)	58.158 (+- 0.399)	554.831 (+- 5.263)
RFoT (Component, CP-APR)	Semi-supervised	0.940 (+-0.016)	0.941 (+-0.016)	0.940 (+-0.016)	93.220 (+- 1.534)	880.700 (+- 21.192)
RFoT (MS, CP-APR)	Semi-supervised	0.793 (+-0.008)	0.805 (+-0.007)	0.797 (+-0.008)	54.218 (+- 1.646)	1582.156 (+- 20.056)
LightGBM	Supervised	0.871 (+-0.005)	0.871 (+-0.005)	0.871 (+-0.005)	NA	**78.595** (+- 6.040)
XGBoost	Supervised	0.873 (+-0.005)	0.874 (+-0.005)	0.873 (+-0.006)	NA	93.805 (+- 2.752)
XGBoost+SelfTrain	Semi-supervised	0.872 (+-0.006)	0.873 (+-0.006)	0.873 (+-0.006)	NA	87.410 (+- 6.813)

4.4 Baseline Comparisons

We compare RFoT with CP-ALS and CP-APR decomposition, using MS and Component clustering, against baseline models *XGBoost*, *LightGBM*, and *XGBoost+SelfTrain*. For each RFoT experiment, we set the hyper-parameters to be same as the Section 4.3.4 except the rank is fixed to $rank = 2$. For CP-APR we use 16 parallel jobs to decompose each random tensor using GPUs, while for CP-ALS is decomposed with 50 parallel jobs on CPUs. We tune the baseline models using a popular Python package *Optuna* [70]. *XGBoost* and *LightGBM* tuned with 3-fold stratified cross-validation and 50 trials to identify the optimal hyper-parameters. The tuning settings, or search space for the optimal hyper-parameters, listed below are the same from our prior work on semi-supervised malware family classification [69].

For *LightGBM*, we used 250 maximum number of iterations, *gbdt* boosting type, and objective function *binary_logloss*. The following hyper-parameters were tuned (ranges are shown in parenthesis): *min_data_in_leaf* (5-100 in log scale), *max_depth* (2-7), *bagging_freq* (0-5), *bagging_fraction* (.5-1.0), *learning_rate* (.001-.1 in log scale), and *feature_fraction* (.1-.7).

As for *XGBoost*, we set the maximum boosting rounds to 250 and use the *binary-hinge* objective function. The following hyper-parameters were tuned (ranges are again shown in parenthesis): *max_depth* (2-10), *eta* (.003-0.5 in log scale), *subsample* (.2-.7), *rounds* (10-300), *colsample_bytree* (.3-1.0), *colsample_bylevel* (.5-1.0), and *lambda* (.1-2.0). We use the same tuned hyper-parameters for *XGBoost* for the *XGBoost+SelfTrain* baseline model.

In Table 4.1, we compare RFoT to baseline models with F1, Precision, Recall, and computation time in seconds. We also show the percent of abstaining predictions for RFoT. From the table, it can be seen that each RFoT model outperforms every other baseline model. RFoT with Component clustering and CP-ALS tensor decomposition yield the highest F1 score of 0.968. However, this model also gives a high abstaining prediction of 75.70%. Therefore, the ideal model choice is RFoT with MS clustering and CP-ALS tensor decomposition with still a high F1 score of 0.91 and a lower abstaining prediction of 58%. CP-APR with component clustering also yield high performance with an F1 score of 0.94; however, the percent of abstaining predictions was the highest with 93.22%. Differently, RFoT with CP-APR decomposition and MS clustering yield the lowest F1 score of 0.79. The fastest model in our comparison was *LightGBM* with 78.59 seconds. These results indicate that RFoT is an ideal model to obtain precise malware detection results by compromising to a lower number of samples that can be predicted due to the abstaining predictions. It is also worth noting that RFoT, a semi-supervised solution, outperforms supervised models with a potential added benefit of better generalizability to novel malware.

4.4.1 Labeled Malware Data Scarcity Experiment

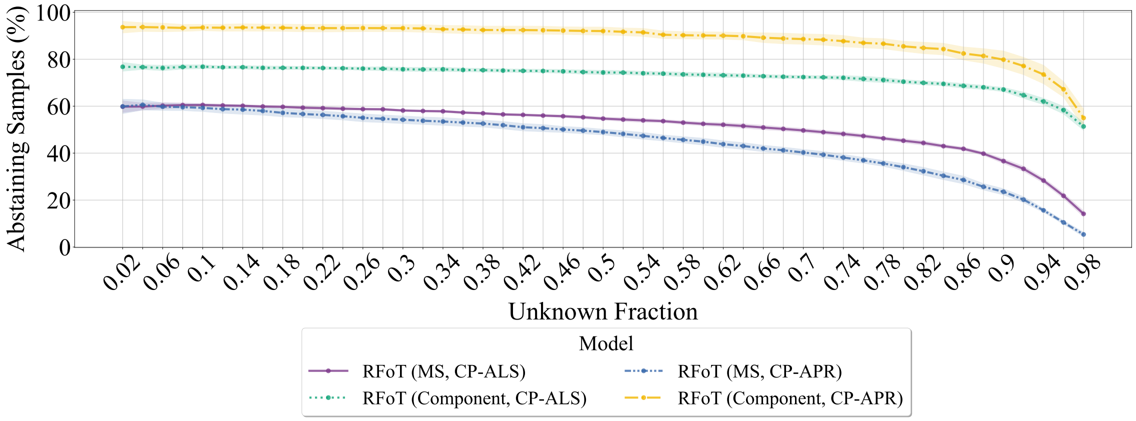

Figure 4.13: Abstaining percentage is shown for different values of the unknown fraction.

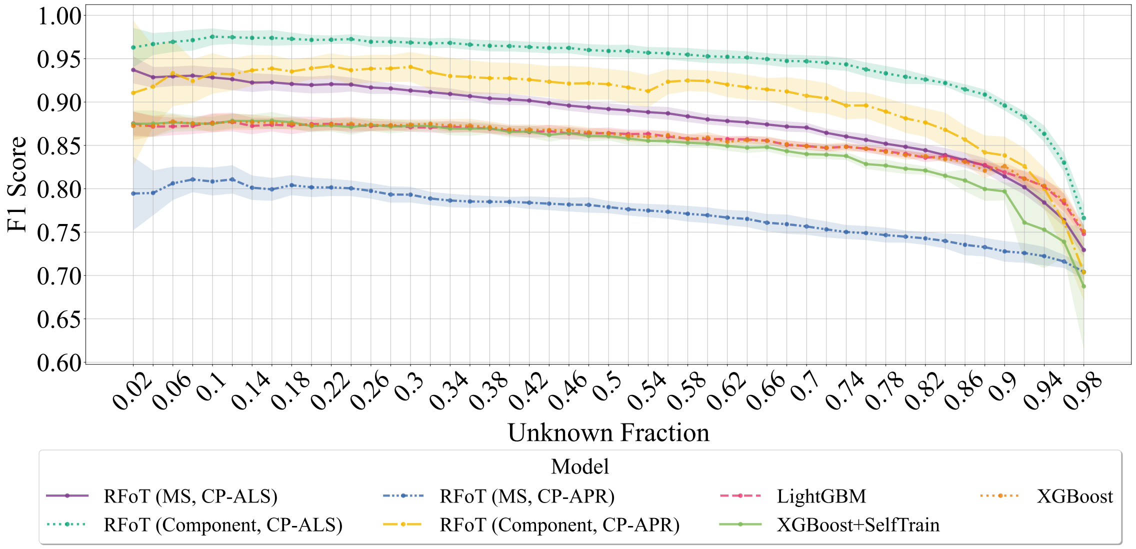

Figure 4.14: F1 score is shown for different values of the unknown fraction.

Compared to other ML fields, acquiring labeled malware data is time-consuming and expensive [8]. This is especially problematic because the popular supervised ML solutions used to detect malware often need a large quantity of labeled data to yield good performance. In addition, Raff et al. emphasize that semi-supervised solutions within the ML-based solutions to Windows malware classification have not received enough attention, despite the potential benefits of better generalizability to novel malware and achieving higher performance when used with low-quantity of labeled data [8]. To this end, we test our semi-supervised solution's performance with a dropping quantity of labeled data and compare it to the supervised and semi-supervised baseline models.

We range the fraction of unknown specimens between 0.02 and 0.98 with the step size of 0.02. The fraction of unknown samples θ means that the proportion of the known samples would be $1 - \theta$. For the supervised baseline models, the fraction of the unknown samples θ is equivalent to the size of the test set, while the fraction of the known samples determines the training set size. The baseline models are also tuned in this experiment as described in Section 4.4.

In Figure 4.13, we can see that the percent of abstaining predictions for RFoT with CP-ALS and CP-APR, and with MS and Component clustering. CP-APR with Component clustering has the highest number of abstaining predictions, and CP-APR with MS clustering has the lowest. At the same time, Figure 4.14 shows that CP-APR with MS clustering has the lowest performance. This indicates that CP-APR is not able to find meaningful patterns that separate the classes within each latent factor. However, CP-APR with Component clustering yields high performance results, with a trade-off of high number of abstaining predictions. CP-ALS with MS and Component clustering has lower percent of abstaining predictions with the higher F1 scores. As the percent of the fraction of the unknown specimens increases, the percent of the abstaining predictions first remain stable and then begins to rapidly drop, specifically after an around unknown fraction of 0.7, as shown in

Figure 4.13. This drop could be happening due to the cluster uniformity score failing to filter out noisy clusters due to now them being represented by known specimens from the same class, as the other known specimens which revealed the bad uniformity are lost with the increasing unknown fraction.

It can be seen in Figure 4.13 that each of our baseline models yield a similar performance as the fraction of unknown specimens increase, except when *XGBoost+SleftTrain* gain a high performance decline after around unknown fraction of 0.74. RFoT with CP-ALS and MS clustering, and CP-APR with Component clustering outperforms each baseline until *XGBoost* and *LightGBM* begins to outperform RFoT with CP-ALS and MS clustering after the unknown fraction of 0.86, and RFoT with CP-APR and Component clustering after unknown fraction of 0.94. Since the abstaining predictions help in maintaining model performance, RFoT based on CP-ALS with Component clustering outperforms each of the baseline model regardless the fraction of unknown specimens.

We have shown the performance of RFoT with different hyper-parameter settings then compared it to the tuned baseline models which prior work used to report state-of-the-art malware detection results. Our results revealed RFoT, a semi-supervised solution, has superior capability when detecting malware compared to the baseline models, including the supervised algorithms. Our experiments also showed that RFoT can yield higher accuracy when detecting malware as the percent of known samples drops. We next list possible areas of future work before concluding the thesis.

Chapter 5: Future Work and Conclusions

5.1 Future Work

In this thesis, we used CP-ALS and CP-APR tensor decomposition algorithms with MS and Component clustering to test the capabilities of RFoT. Future work includes using other tensor decomposition methods and clustering approaches to investigate if the performance of RFoT can be improved. We also note that we have experimented with using RFoT on other datasets and for multi-class classification which has returned promising results indicating a good generalizability of our algorithm to other problems. For instance, we find that RFoT was able to perform accurate multi-class classification using the IRIS dataset and perform accurate classification on a TF-IDS features matrix of documents from 2 classes obtained using the 20-Newsgroup dataset. We provide these examples in our repository for RFoT[1]. Future work can perform a detailed investigation into using RFoT for other data, and also for multi-class classification. For instance, the EMBER-2018 dataset includes AV-Class labels for the malware samples. Future work can use these samples to test if RFoT can be used to classify malware families. In addition, this work uti-

---

[1] Examples for when using RFoT on other problems is available at https://github.com/MaksimEkin/RFoT/tree/main/examples

lized only a fraction of the features available in the EMBER-2018 dataset. Other, potentially more detailed, features such as byte-histogram can be used to form the tensors, which could potentially improve the specificity of our model. We also leave this to future work.

5.2 Conclusions

We introduced a semi-supervised method, named RFoT, that leverages tensor decomposition for classifying malware and benign ware. Tensor decomposition can discover meaningful latent patterns that can be captured via clustering methods. These patterns distinguish malware and benign-ware and enable us to use known samples as a reference point to vote on class labels of the unknown specimens. Because the information extracted with tensor decomposition depends on the configuration of the tensor, we formulate a model that creates an ensemble of random configurations and makes use of the decisions made by the majority of the population in the ensemble via max-vote to make the final class prediction. We showed that our semi-supervised method yields more precise classification results compared to the baseline supervised and semi-supervised ML models, with a trade-off on being able to predict a lower number of samples due to the abstaining predictions.

Bibliography

[1] K. Bissell and L. Ponemon. The cost of cybercrime. Technical report, Accenture, Ponemon Institute, 2019.

[2] Cost of a data breach report. Technical report, IBM, 2019.

[3] The Independent IT Security Institute. Malware statistics & trends report: Av-test, Feb 2022.

[4] State of malware report. Technical report, Malwarebytes Labs, February 2020.

[5] Data breach investigations report 2021. Technical report, Verizon, 2021.

[6] Michael Sikorski and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 2012.

[7] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole exe. In AAAI Workshops, 2018.

[8] Edward Raff and C. Nicholas. A survey of machine learning methods and challenges for windows malware classification. ArXiv, abs/2006.09271, 2020.

[9] Casey Battaglino, G. Ballard, and T. Kolda. A practical randomized CP tensor decomposition. SIAM J. Matrix Anal. Appl., 39:876–901, 2018.

[10] Brett W. Bader and Tamara G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. ACM Transactions on Mathematical Software, 32(4):635–653, December 2006.

[11] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. SIAM Review, 51(3):455–500, September 2009.

[12] Eric C. Chi and Tamara G. Kolda. On tensors, sparsity, and nonnegative factorizations. SIAM J. Matrix Anal. Appl., 33:1272–1299, 2012.

[13] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Transactions on Information Theory, 21(1):32–40, 1975.

[14] Kuo-Lung Wu and Miin-Shen Yang. Mean shift-based clustering. Pattern Recogn., 40(11):3035–3052, nov 2007.

[15] Xin Jin and Jiawei Han. Mean Shift, pages 806–808. Springer US, Boston, MA, 2017.

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

[17] H. S. Anderson and P. Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. ArXiv e-prints, April 2018.

[18] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[19] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pages 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[20] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics, ACL '95, pages 189–196, USA, 1995. Association for Computational Linguistics.

[21] M. E. Eren, C. Nicholas, R. McDonald, and C. Hamer. Random Forest of Tensors (RFoT), 2021. Presented at the 12th Annual Malware Technical Exchange Meeting (MTEM), Sandia National Laboratories, Online, 2021.

[22] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pages 108–122, 2013.

[23] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 3.0-dev. Available online, August 2017.

[24] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer,

Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020.

[25] Samantha Hansen, Todd Plantenga, and Tamara G. Kolda. Newton-based optimization for Kullback-Leibler nonnegative tensor factorization. Optimization Methods and Software, 30(5):1002–1029, April 2015.

[26] M. E. Eren, J. S. Moore, E. Skau, M. Bhattarai, G. Chennupati, and B. S. Alexandrov. pycp_apr. https://github.com/lanl/pyCP_APR, 2021.

[27] Vinayakumar R. and Soman K.P. Deepmalnet: Evaluating shallow and deep networks for static pe malware detection. ICT Express, 4(4):255–258, 2018.

[28] Fabian H. Fonseca A, Serena Ferracci, Federico Palmaro, Luca Iocchi, Daniele Nardi, and Luisa Franchina. Static analysis of pe files using neural network techniques for a pocket tool. In 2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), pages 01–06, 2021.

[29] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. IEEE Access, 7:46717–46738, 2019.

[30] Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic malware classification and new malware detection using machine learning. Frontiers of Information Technology & Electronic Engineering, 18(9):1336–1347, 2017.

[31] Jinpei Yan, Yong Qi, and Qifan Rao. Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, 2018, 2018.

[32] Barath Narayanan Narayanan and Venkata Salini Priyamvada Davuluru. Ensemble malware classification system using deep neural networks. Electronics, 9(5), 2020.

[33] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. Computers & Security, 92:101740, 2020.

[34] Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. A novel method for malware detection on ml-based visualization technique. Computers & Security, 89:101682, 2020.

[35] S. Abijah Roseline, A. D. Sasisri, S. Geetha, and C. Balasubramanian. Towards efficient malware detection and classification using multilayered random forest

ensemble technique. In 2019 International Carnahan Conference on Security Technology (ICCST), pages 1–6, 2019.

[36] Salma Abdelmonem, Shahd Seddik, Rania El-Sayed, and Ahmed S. Kaseb. Enhancing image-based malware classification using semi-supervised learning. In 2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), pages 125–128, 2021.

[37] Shuwei Wang, Qiuyun Wang, Zhengwei Jiang, Xuren Wang, and Rongqi Jing. A weak coupling of semi-supervised learning with generative adversarial networks for malware classification. In 2020 25th International Conference on Pattern Recognition (ICPR), pages 3775–3782, 2021.

[38] Rajesh Kumar and S Geetha. Malware classification using xgboost-gradient boosted decision tree. Advances in Science, Technology and Engineering Systems Journal, 5(5), 2020.

[39] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. Static pe malware detection using gradient boosting decision trees algorithm. In Tran Khanh Dang, Josef Küng, Roland Wagner, Nam Thoai, and Makoto Takizawa, editors, Future Data and Security Engineering, pages 228–236, Cham, 2018. Springer International Publishing.

[40] Panpan Qi, Wei Wang, Lei Zhu, and See Kiong Ng. Unsupervised Domain Adaptation for Static Malware Detection Based on Gradient Boosting Trees, pages 1457–1466. Association for Computing Machinery, New York, NY, USA, 2021.

[41] Venkata Atluri. Malware classification of portable executables using tree-based ensemble machine learning. In 2019 SoutheastCon, pages 1–6. IEEE, 2019.

[42] Fauzan Hikmah Ramadhan, Vera Suryani, and Satria Mandala. Analysis study of malware classification portable executable using hybrid machine learning. In 2021 International Conference on Intelligent Cybernetics Technology Applications (ICICyTA), pages 86–91, 2021.

[43] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 3422–3426, 2013.

[44] Nureni Ayofe Azeez, Oluwanifise Ebunoluwa Odufuwa, Sanjay Misra, Jonathan Oluranti, and Robertas Damaševičius. Windows pe malware detection using ensemble learning. In Informatics, volume 8, page 10. Multidisciplinary Digital Publishing Institute, 2021.

[45] Deepak Gupta and Rinkle Rani. Improving malware detection using big data and ensemble learning. Computers & Electrical Engineering, 86:106729, 2020.

[46] Yanfang Ye, Tao Li, Yong Chen, and Qingshan Jiang. Automatic malware categorization using cluster ensemble. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 95–104, New York, NY, USA, 2010. Association for Computing Machinery.

[47] Yunan Zhang, Chenghao Rong, Qingjia Huang, Yang Wu, Zeming Yang, and Jianguo Jiang. Based on multi-features and clustering ensemble method for automatic malware categorization. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 73–82, 2017.

[48] Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1357–1365, New York, NY, USA, 2013. Association for Computing Machinery.

[49] Edward Raff, Charles K. Nicholas, and Mark McLean. A new burrows wheeler transform markov distance. In *AAAI*, 2020.

[50] Paul Irofti and Andra BfÉltoiu. Malware identification with dictionary learning. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019.

[51] Danai Koutra, Evangelos E. Papalexakis, and Christos Faloutsos. Tensorsplat: Spotting latent anomalies in time. In *2012 16th Panhellenic Conference on Informatics*, pages 144–149, 2012.

[52] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 203–210, 2011.

[53] David Bruns-Smith, Muthu Manikandan Baskaran, James R. Ezick, Thomas Henretty, and Richard A. Lethin. Cyber security through multidimensional data decompositions. *2016 Cybersecurity Symposium (CYBERSEC)*, pages 59–67, 2016.

[54] M. E. Eren, J. S. Moore, and B. S. Alexandrov. Multi-dimensional anomalous entity detection via poisson tensor factorization. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6, 2020.

[55] M. E. Eren, J. S. Moore, E. W. Skau, E. A. Moore, M. Bhattarai, G. Chennupati, and B. S. Alexandrov. General-purpose unsupervised cyber anomaly detection via non-negative tensor factorization. *Digital Threats: Research and Practice*, 2022.

[56] James Ezick, Tom Henretty, Muthu Baskaran, Richard Lethin, John Feo, Tai-Ching Tuan, Christopher Coley, Leslie Leonard, Rajeev Agrawal, Ben Parsons, and William Glodek. Combining tensor decompositions and graph analytics to provide cyber situational awareness at hpc scale. In 2019 IEEE High Performance Extreme Computing Conference (HPEC), pages 1–7, 2019.

[57] Hideaki Kanehara, Yuma Murakami, Jumpei Shimamura, Takeshi Takahashi, Daisuke Inoue, and Noboru Murata. Real-time botnet detection using nonnegative tucker decomposition. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, pages 1337–1344, New York, NY, USA, 2019. Association for Computing Machinery.

[58] Hadi Fanaee-T, Márcia D. B. Oliveira, João Gama, Simon Malinowski, and Ricardo Morla. Event and anomaly detection using tucker3 decomposition. ArXiv, abs/1406.3266, 2014.

[59] Kun Xie, Xiaocan Li, Xin Wang, Gaogang Xie, Jigang Wen, Jiannong Cao, and Dafang Zhang. Fast tensor factorization for accurate internet anomaly detection. IEEE/ACM Transactions on Networking, 25(6):3794–3807, 2017.

[60] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In KDD '06, 2006.

[61] Ilia Kisil, Ahmad Moniri, and Danilo P. Mandic. Tensor ensemble learning for multidimensional data. In 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pages 1358–1362, 2018.

[62] Jieqiong Hou, Minhui Xue, and Haifeng Qian. Unleash the power for tensor: A hybrid malware detection system using ensemble classifiers. In 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), pages 1130–1137, 2017.

[63] David Hong, Tamara G. Kolda, and Jed A. Duersch. Generalized canonical polyadic tensor decomposition. ArXiv, abs/1808.07452, 2020.

[64] Yang Qi, Pierre Comon, and Lek-Heng Lim. Semialgebraic geometry of nonnegative tensor rank. SIAM J. Matrix Anal. Appl., 37:1556–1580, 2016.

[65] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. ArXiv, abs/1711.10781, 2017.

[66] R. Vangara, E. Skau, G. Chennupati, H. Djidjev, T. Tierney, J. P. Smith, M. Bhattarai, V. G. Stanev, and B. S. Alexandrov. Semantic nonnegative matrix factorization with automatic model determination for topic modeling. In 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 328–335, 2020.

[67] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[68] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, Proceedings of the 9th Python in Science Conference, pages 56 – 61, 2010.

[69] M. E. Eren, M. Bhattarai, R. J. Joyce, E. Raff, C. Nicholas, and B. Alexandrov. Semi-supervised classification of malware families via hierarchical non-negative matrix factorization with automatic model determination. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2021. LA-UR-21-29919.

[70] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pages 2623–2631, 2019.