Semi-supervised Classification of Malware Families Under Extreme Class Imbalance via Hierarchical Non-Negative Matrix Factorization with Automatic Model Selection

MAKSIM E. EREN, Advanced Research in Cyber Systems, LANL, USA
MANISH BHATTARAI, Theoretical Division, LANL, USA
ROBERT J. JOYCE, Machine Learning Research Group, Booz Allen Hamilton, USA
EDWARD RAFF, Machine Learning Research Group, Booz Allen Hamilton, USA
CHARLES NICHOLAS, Department of Computer Science and Electrical Engineering, UMBC, USA
BOIAN S. ALEXANDROV, Theoretical Division, LANL, USA

Identification of the family to which a malware specimen belongs is essential in understanding the behavior of the malware and developing mitigation strategies. Solutions proposed by prior work, however, are often not practicable due to the lack of realistic evaluation factors. These factors include learning under class imbalance, the ability to identify new malware, and the cost of production-quality labeled data. In practice, deployed models face prominent, rare, and new malware families. At the same time, obtaining a large quantity of up-to-date labeled malware for training a model can be expensive. In this paper, we address these problems and propose a novel hierarchical semi-supervised algorithm, which we call the *HNMFk Classifier*, that can be used in the early stages of the malware family labeling process. Our method is based on non-negative matrix factorization with automatic model selection, that is, with an estimation of the number of clusters. With *HNMFk Classifier*, we exploit the hierarchical structure of the malware data together with a semi-supervised setup, which enables us to classify malware families under conditions of extreme class imbalance. Our solution can perform abstaining predictions, or rejection option, which yields promising results in the identification of novel malware families and helps with maintaining the performance of the model when a low quantity of labeled data is used. We perform bulk classification of nearly 2,900 both rare and prominent malware families, through static analysis, using nearly 388,000 samples from the EMBER-2018 corpus. In our experiments, we surpass both supervised and semi-supervised baseline models with an F1 score of 0.80.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Semi-supervised learning settings**; **Non-negative matrix factorization**; **Topic modeling**.

Additional Key Words and Phrases: malware, malware families, non-negative matrix factorization, semi-supervised, hierarchical, model selection, class imbalance, abstaining prediction, reject-option

1 INTRODUCTION

The objective of malware detection is to identify a given file as benign or malicious, typically by using its run-time behavior (dynamic malware analysis) and/or static information (static malware analysis). In contrast to malware detection, malware family classification assumes that any given sample is already known to be malicious, and we want to know which family it belongs to [58]. New malware samples are created regularly by threat actors by

Authors' addresses: Maksim E. Eren, Advanced Research in Cyber Systems, LANL, USA, maksim@lanl.gov; Manish Bhattarai, Theoretical Division, LANL, USA, ceodspspectrum@lanl.gov; Robert J. Joyce, Machine Learning Research Group, Booz Allen Hamilton, USA, ceodspspectrum@lanl.gov; Edward Raff, Machine Learning Research Group, Booz Allen Hamilton, USA, raff_edward@bah.com; Charles Nicholas, Department of Computer Science and Electrical Engineering, UMBC, USA, nicholas@umbc.edu; Boian S. Alexandrov, Theoretical Division, LANL, USA, boian@lanl.gov.

2471-2566/2023/9-ART $15.00
https://doi.org/10.1145/3624567

various techniques, which create new versions of already existing malware specimens with identical functionality [58]. Malware analysts regularly go through large quantities of malware samples to understand if a new specimen in fact belongs to a previously known malware family. Classifying a new malware sample into a family can reduce the number of files analysts need to examine, and aid in understanding the behavior of the malware; this is in turn helpful for estimating the severity of the threat, developing mitigation strategies, and building datasets [58]. The tools that can aid in malware detection and classification are especially significant now as recent reports point out that malware is one of the most frequent and costly cyber threats [15].

Approximately half a million new malware specimens are reported daily, which drives the increased utilization of Machine learning (ML) based automated security systems to combat malware [41, 49, 55, 56, 65]. However, the adoption of ML-based solutions against malware threats has been relatively slow despite the cost savings [36]. Shortcomings in the existing solutions are perhaps contributing to this problem. The majority of prior research for malware family classification, over the past two decades, has not sufficiently accounted for core evaluation criteria in their work including learning under class imbalance, ability to identify new malware, and the cost of production-quality labeled data [54, 58]. For example, the majority of ML solutions for malware family classification are unrealistically limited to identifying the top most populous families. This results in reports of excellent performance on evaluation metrics that do not generalize to the real world, limited as they have been to the analysis of *"easy"* malware. At the same time, semi-supervised learning in the malware classification field has not been widely explored despite its potential benefits [58]. With the ever-growing quantity of malware, attacks, and their complexities there is an urgent need to improve existing solutions and their operational architectures to drive the increased adaption of ML-based solutions.

In this work, we introduce a novel semi-supervised algorithm, named Hierarchical Non-Negative Matrix Factorization with automatic model selection Classifier (or *HNMFk Classifier*). The HNMFk Classifier classifies Windows Portable Executable (PE) format malware specimens (e.g. from the EMBER-2018 dataset) into families using static malware analysis-based features [9]. Our method performs bulk classification where the known samples are used as a reference against the unknown specimens when performing hierarchical clustering, resulting in a model with only an inference process (i.e. no training). Therefore, in comparison to the traditional ML models which have separate training (slow) and prediction (fast) steps, our solution can be used outside the real-time environments, such as early stages in the labeling process of the malware. *HNMFk Classifier* performs hierarchical clustering using Non-Negative Matrix Factorization (NMF) with automatic model selection (*NMFk*) [3, 4, 6, 7, 23–25, 53], which helps us determine whatever hierarchical structure exists among the malware specimens. With a semi-supervised setting, we obtain the ability to perform abstaining predictions (i.e. predicting "I do not know"), in addition to answering either "Yes" or "No". Specifically, our model incorporates a reject option, where it abstains from making a prediction (reject). Abstaining predictions aid in detecting novel malware, reduce the need to include all malware families during factorization to achieve good generalizability to new malware, and help our model to maintain its performance with a low quantity of labeled data.

In our experiments, we first use a small subset of the dataset (in a setup that does not reflect the real world) to understand the effects of using different hyper-parameters in our model, conduct ablation studies, and to observe the performance of our model with a decreasing quantity of labeled data. During our (more realistic) larger scale experiments, we use 2,898 classes of malware families (numbering more than 388,000 samples) with extreme class imbalance, and while including novel unknown malware samples during classification. Our method surpasses the supervised baseline models *XGBoost* and *LightGBM* [21, 42]. We further extend these baselines with the *SelfTrain* algorithm to create strong semi-supervised models, which our approach still outperforms [73]. We also achieve better classification results compared to our Multilayer Perceptron (MLP) baseline [33]. To the best of our knowledge, we are the first to perform malware family classification over the EMBER-2018 corpus under realistic conditions such as the inclusion of the rare and novel families during our experiments, and our target number of family classes is around 29 times more than the previous work with the largest number of classes [35]. Our contributions include:

- Introducing a novel semi-supervised hierarchical bulk classifier, the *HNMFk Classifier*, that can assist analysts early in the malware family labeling process.
- Identifying Windows malware families using static malware analysis-based features, specifically using malware meta-data and PE header features, under extreme class imbalance conditions.
- Utilizing abstaining prediction to enable our model to help identification of novel malware families, and maintain its accuracy as the amount of labeled data decreases.
- Achieving higher F1 scores compared to the baseline supervised and semi-supervised learners which prior work used to report their benchmarks when classifying malware families in the EMBER-2018 corpus.

The remainder of the paper is organized as follows: we provide a summary of related work in Section 2. Section 3 includes a description of NMF (Section 3.1), automatic model selection with *NMFk* (Section 3.2), and hierarchical NMF (Section 3.3). We then introduce our *HNMFk Classifier* in Section 3.4. Section 4 describes the dataset and the features used in our experiments, pre-processing of the features, and the preparation of the experiments. Section 5 showcases our experimental results including a performance analysis over a subset of the data and how our model preserves accuracy under the low quantity of labeled data in Section 5.1. Our results when classifying the malware families under realistic conditions, and comparison to the baseline models, are shown in Section 5.2. We justify the parts of our methodology with ablation studies in Section 5.3. Before concluding, we list potential areas of future work to explore in Section 6.

2 RELATED WORK

Malware classification is a challenging task, and the quantity and complexity of malware continues to increase rapidly. This makes the ML-based malware classification an important field of study. Raff et al. surveys over 200 research articles on ML-based malware analysis [58]. This survey of the field emphasizes that the standard ML model evaluation technique, where the dataset containing malware families are divided into training and test sets, is flawed when it comes to the malware family classification problem in the real-life case, since previously unseen malware families will continue to appear. To this end, they recommend that the ability to perform abstaining prediction can assist analysts in identifying novel malware. However, prior work has not widely studied this open problem area for malware classification. In our experiments, we evaluate the performance of our solution by including a set of malware families that were not present in the known set. Additionally, Raff et al. discuss the challenges in malware data gathering, and the expensive and time-consuming process of file labeling. Their survey found that semi-supervised solutions are not yet fully explored, although they can help when faced with only a small quantity of labeled data. In support of this finding, we show that our solution continues to maintain its performance with the decreasing amount of labeled malware in our small-scale experiment in Section 5.1. Finally, Raff et al. also point out the relatively small amount of prior work on the problem of class imbalance. This issue was also emphasized by Nguyen et al. since much prior work has unrealistically evaluated their solutions over the top most populous malware families [54]. Our study addresses this problem by including both the rare and prominent classes of malware families during the large-scale experiment in Section 5.2.

Several previous works have looked at malware family classification, however, they tend to use only the most common malware families, did not consider novel malware families, or used manually balanced datasets when reporting their results [1, 11, 27, 32, 40, 47, 63, 71, 75]. In contrast, when comparing to the baseline models, we report our results when classifying specimens belonging to the whole ensemble of malware families present in the EMBER-2018 dataset with an imbalanced setup which also includes novel unknown specimens (Section 5.2). This setup allows our results to be more like what malware analysts would see in the real world. Several prior works also considered class imbalance, however, they still targeted a small number of top malware families, and rare specimens are mapped to a single *"others"* class [47, 50]. To the best of our knowledge, the most realistic and the largest malware family classification work was done by Huang et al. [35], which targeted 100 classes where two of the classes include the one for benign samples and another for the rare specimens. This type of setup, although it considers class imbalance, limits the classification capabilities to only a handful of malware families. In contrast, we do not map the rare specimens into a single class, but rather recognize all 2,898 malware families as individual classes. Furthermore, supervised methods used in prior work often poorly generalize to rare specimens as also pointed out by Loi et al [47]. Loi et al. reports that their false positives are heavily represented by the families collected within the *"others"* class due to the supervised method's inability to learn the patterns of these families from a rare number of specimens. We use a semi-supervised approach, which has an added benefit of improved generalizability and ability to work with a low quantity of labeled data compared to the supervised models. A number of these prior works did consider benign-ware as a class in their analysis [47, 50, 71], but we assume the samples are already known to be malware and perform only malware family classification. We summarize the mentioned prior work and show how they compare to our research in Table 1.

Table 1. The comparison of prior and our work in dataset size, number of classes, consideration of imbalanced data and novel malware families, and the method used. *Custom* refers to the proprietary datasets, or the custom build datasets by the authors.

Reference	**Dataset(s)**	**Dataset Size**	**Num. Classes**	**Imbalanced Data**	**Novel Malware**	**Method**
Ours	EMBER-2018 [9]	388k	2,898	✓	✓	Semi-supervised
[35]	*Custom*	6.5m	100	✓	–	Supervised
[40]	Drebin [10]	5k	40	–	–	Supervised
[71]	Malimg [52] & *Custom*	9k & 10k	25 & 10	–	–	Supervised
[11]	*Custom*	10k	14	✓	–	Unsupervised
[75]	EMBER-2018 [9]	750k	21	–	–	Supervised
[47]	EMBER-2017 [9]	500k	21	✓	–	Supervised
[63]	VirusShare [26]	2.7k	12	–	–	Supervised
[1]	Malimg [52]	21k	9	–	–	Supervised
[50]	*Custom*	115k	8	✓	✓	Supervised
[32]	*Custom*	31k	5	–	–	Supervised

Non-negative Matrix Factorization, or NMF, has also been applied to the malware/benign-ware classification problem. Ling et al. derive similarity scores of structural patterns extracted with NMF to detect metamorphic malware (malware with the capability to modify its code during run-time) using static analysis features [46]. In their experiments, they choose a fixed number of components for NMF where the number of components k is selected as $k(n+m) < nm$. A single application of NMF misses the patterns hidden in malware sub-groups, and using a fixed number of components can result in missing important information (under-fitting) or including noise (over-fitting) in the results. Unlike Ling et al., we perform malware family classification by applying hierarchical NMF to discover the sub-groups and utilize *NMFk* as a heuristic to determine the number of components or clusters. Prior work outside the malware analysis field has demonstrated that hierarchical NMF can be used to achieve good clustering of the data [29, 66]. Gillis et al. show that using rank-two factorization at each step (i.e. split the data into two at each stage, $k = 2$) yields good clustering results when applied with hierarchical NMF [29]. We use hierarchical rank-two NMF in our ablation studies and show that estimating the number of components via *NMFk* produces better classification results, although extracting two clusters at each factorization does yield good classification results that surpass our baseline models.

3 METHODS

Our work draws on prior advances in standard and hierarchical NMF methods, and automatic model selection. In this section, we give a brief summary for each of these methods, then we introduce our *HNMFk* Classifier. The summary of the notations used throughout the paper is provided in Table 2.

Table 2. Summary of the notation styles used in the paper.

Notation	Description
x	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\mathfrak{X}$	Tensor
$\mathbf{x}_i$	ith element in the vector
$\mathbf{X}_{ij}$	Entry located on row i and column j
$\mathbf{X}_{i:}$	ith row
$\mathbf{X}_{:j}$	jth column
$\mathfrak{X}_{::i}$	ith slice along the third dimension
$\mathfrak{X}_{:::i}$	ith slice along the fourth dimension
$\mathfrak{X}^{name}$	Superscript *name* used as an identifier
$*$	Dot product

3.1 Non-negative Matrix Factorization (NMF)

NMF is an unsupervised learning method based on a low-rank matrix approximation. NMF represents an observed non-negative matrix, $\mathbf{X} \in \mathbb{R}_+^{n \times m}$, as a product of two (unknown) non-negative matrices, $\mathbf{W} \in \mathbb{R}_+^{n \times k}$, and $\mathbf{H} \in \mathbb{R}_+^{k \times m}$, where usually $k \ll m, n$. Here, n is the number of samples, and m is the number of features. This approximation is performed via non-convex minimization with a given distance, $||...||_{dist}$, constrained by the non-negativity of $\mathbf{W}$ and $\mathbf{H}$: $\min||\mathbf{X}_{ij} - \sum_{s=1}^{k} \mathbf{W}_{is}\mathbf{H}_{sj}||_{dist}$. NMF relies on a generative statistical model predetermined by the choice of the distance $||...||_{dist}$. For example, if the Frobenius norm is chosen as a distance, NMF can be treated as a Gaussian mixture model [28]. If KL-divergence is chosen, we have a generative Poisson model [19], equivalent to latent Dirichlet allocation under uniform Dirichlet prior [22]. In both cases, the number of latent features of the superimposed components is equal to the size of the small dimension k, and NMF minimization is equivalent to the expectation-minimization (EM) algorithm. In this probabilistic interpretation of NMF, the observables are the rows of $\mathbf{X}$ generated by latent variables, the rows of the matrix $\mathbf{W}$, with weights (the basis patterns), represented by the columns of matrix $\mathbf{H}$. Thus, each row $\mathbf{X}_{i:}$ of $\mathbf{X}$ is generated from a probability distribution with mean $\mathbf{X}_{i:} = \sum_{s=1}^{k} \mathbf{H}_{is}\mathbf{W}_{s:}$.

3.2 Automatic Model Selection: NMFk

The NMF minimization requires prior knowledge of the latent dimensionality, k (the number of latent features), which is usually unavailable. It is known that choosing too small a value of k leads to a poor approximation of the observables in $\mathbf{X}$ (*under-fitting*), while if k is chosen to be too large, the extracted features are not easily explainable because they also fit the noise in the data (*over-fitting*). In other words, choosing k is equivalent to estimating the number of parameters of the model, which is a difficult and a well-known problem.

In general, the existing partial solutions of this problem are heuristic. Among these solutions is Automatic Relevance Determination (ARD) [48] which was first modified for Principal Component Analysis [14], and then for NMF [51, 64]. Another approach is based on an assumed stability of the NMF solution, and was proposed to identify the number of stable clusters in the observational matrix $\mathbf{X}$ [18]. A recent model selection technique, called *NMFk* [3], has been successfully used to decompose the largest collection of human cancer genomes [6]. *NMFk* integrates classical NMF-minimization with custom clustering and Silhouette statistics [61], and combines the accuracy of the minimization and robustness/stability of the NMF solutions, when a bootstrap procedure (i.e., generation of a random ensamble of slightly perturbed input matrices) is applied to estimate the number of latent features, see for example, [7]. Recently, *NMFk* was applied to a large number of synthetic datasets with a predetermined number of latent features, and it was demonstrated its superior performance of correctly estimating k in comparison to the other known heuristics [53]. The superior performance of *NMFk* method as a model selection was also demonstrated in identifying mutational genome signatures in a large set of cancer genomes, both in practice [5] and in large set of synthetic cancer genomes with predetermined number of latent features [37]. In addition, it was shown that *NMFk* performs better than spherical k-means and other methods for topic extraction [69]. Our numerical experiments here demonstrate that *NMFk* performs better than the predetermined k=2 case. Therefore, we use *NMFk* as the core factorization method with automatic model selection needed to extract the right clusters of malware, after the NMF dimension reduction. In this work, we are making extensive use of *NMFk*, and for completeness we provide the pseudocode for it in Algorithm 1 and a description of it, as follows:

Algorithm 1 NMFk($\mathbf{X}$, k^{min}, k^{max}, M, $Sill\_thr$ = 0.8)

```
Require: : $\mathbf{X} \in \mathbb{R}_{+}^{n\times m}$ , $k^{min}$, $k^{max}$ , $r$
1:  for $k$ in $k^{min}$ to $k^{max}$ do                                  ▷ Start and end process for NMFk
2:      for $q$ in 1 to $M$ do                                             ▷ Num. of Perturbations on each k
3:          $\mathcal{X}_{::q}$ = Perturb($\mathbf{X}$)                     ▷ Resampling X to create a random ensemble
4:          $\mathcal{W}_{::kq}$,$\mathcal{H}_{::kq}$ = NMF($\mathcal{X}_{::q}$,k)
5:      end for
6:      $\mathcal{W}^{all}$=[$\mathcal{W}_{::k1}$,...,$\mathcal{W}_{::kM}$] and $\mathcal{H}^{all}$=[$\mathcal{H}_{::k1}$,...,$\mathcal{H}_{::kM}$]
7:      $\hat{\mathcal{W}}$, $\hat{\mathcal{H}}$ = customCluster( $\mathcal{W}^{all}$,$\mathcal{H}^{all}$)
8:      $\widetilde{\mathcal{W}}_{::k}$ = medians( $\hat{\mathcal{W}}$)
9:      $\mathcal{H}^{reg}_{::k}$ = NNLS($\mathbf{X}$,$\widetilde{\mathbf{W}}_{::k}$)     ▷ Column-wise regression of H with $\widetilde{\mathbf{W}}$ and column of X
10:     $\mathbf{s}_k$ = clusterStability($\hat{\mathcal{W}}$)
11:     $\mathbf{err_k}$ = reconstructErr($\mathbf{X}$,$\widetilde{\mathbf{W}}_{::k}$ , $\mathbf{H}^{reg}_{::k}$ )   ▷ Column-wise reconstruction error for L-statistics
12: end for
13: $\mathbf{err}^{all}$=[$\mathbf{err}_{k^{min}}$,...,$\mathbf{err}_{k^{max}}$]
14: $k^{opt}$ = PvalueAnalysis($\mathbf{err}^{all}$ ,$k^{min}$,$k^{max}$,$\mathbf{s}_k$,$Sill\_thr$)     ▷ Predicted k value using Wilcoxon
15: return $\widetilde{\mathcal{W}}_{::k^{opt}}$, $\mathcal{H}^{reg}_{::k^{opt}}$, $k^{opt}$
    Ensure: $k = k^{opt}$,$\widetilde{\mathcal{W}}_{::k^{opt}} \in \mathbb{R}_{+}^{n\times k}$ ,$\mathcal{H}^{reg}_{::k^{opt}} \in \mathbb{R}_{+}^{k\times m}$ , $\mathbf{X} = \widetilde{\mathcal{W}}_{::k^{opt}}\ \mathcal{H}^{reg}_{::k^{opt}}$
```

(1) *Resampling*: Based on the observable matrix, $\mathbf{X}$, *NMFk* creates an ensemble of M random matrices, $[\mathcal{X}_{::q}]_{q=1,...,M}$, with means equal to the original matrix $\mathbf{X}$. Each one of these random matrices $\mathcal{X}_{::q}$ is generated by perturbing the elements of $\mathbf{X}$ by a small uniform noise, such that: $\mathcal{X}_{ijq} = \mathbf{X}_{ij} + \delta$, for each $q = 1, ..., M$, where δ is the small error.
(2) *NMF minimization*: We use the Frobenius norm-based multiplicative updates (MU) algorithm [44] to explore different numbers of latent features, k, in an interval $[k^{min}, k^{max}]$, for each one of the generated M random matrices.

(3) *Custom clustering:* For each $k \in [k^{min}, k^{max}]$, NMF minimizations of the M random matrices, $[\mathcal{X}_{::q}]_{q=1,...,M}$, results in M pairs $[\mathcal{W}_{::kq}; \mathcal{H}_{::kq}]_{q=1,...,M}$. Further, *NMFk* clusters the set of the $M * k$ latent features, the columns of $\mathcal{W}_{::kq}$. The *NMFk* custom clustering is similar to k-means, but it holds in each one of the clusters exactly one column from each of the M NMF solutions. This constraint is needed since each NMF minimization gives exactly one solution $\mathcal{W}_{::kq}$ with the same number of columns, k. In the clustering, the similarity between the columns is measured by the cosine similarity metric.

(4) *Robust* $\mathbf{W}$ *and* $\mathbf{H}$ *for each* k*:* The medians of the clusters, $\widetilde{\mathcal{W}}_{::k}$, are the robust solution for each explored k. The corresponding mixing coefficients $\mathcal{H}^{reg}_{::k}$ are calculated by regression of $\mathbf{X}$ on $\widetilde{\mathcal{W}}_{::k}$.

(5) *Cluster stability via Silhouette statistics: NMFk* explores the stability of the obtained clusters, for each k, by calculating their Silhouettes [61]. Silhouette statistics quantify the cohesion and separability of the clusters. The Silhouette values range between $[-1, 1]$, where -1 means an unstable cluster, while $+1$ means perfect stability.

(6) *Reconstruction error:* Another metric *NMFk* uses is the relative reconstruction error, $R = ||\mathbf{X} - \mathcal{X}^{rec}_{::k}||/||\mathbf{X}||$, where $\mathcal{X}^{rec}_{::k} = \widetilde{\mathcal{W}}_{::k} * \mathcal{H}^{reg}_{::k}$, which measures the accuracy of the reproduction of initial data by a given solution and the number of latent features k.

(7) *L-statistics: NMFk* uses L-statistics [68] to automatically estimate the number of latent features. To calculate L-statistics for each k, *NMFk* records the distributions of the column reconstruction errors, $\mathbf{e}_i = \|\mathbf{X}_{:j} - \mathcal{X}^{rec}_{:jk}\|/\|\mathbf{X}_{:j}\|$; $j = 1, ..., m$. L-statistics compares the distributions of column errors for different k by a two-sided Wilcoxon rank-sum test [34], which evaluates whether two samples are taken from the same population.

(8) *NMFk final solution:* The number of latent features, k^{opt}, is determined as the maximum number of stable clusters corresponding to a good accuracy of the reconstruction. The Wilcoxon rank-sum test determines the p-value of the given k^{opt}. *NMFk* is "looking" for a distribution of the column errors such that the next distributions (each one with bigger k) are statistically the same, and the model is fitting the noise. The L-statistics used in conjunction with the condition that the *minimum* Silhouette be greater than 0.80. The threshold of 0.80 is selected to place the predicted k^{opt} prior to a steep decline in the *minimum* Silhouette. The corresponding $\widetilde{\mathcal{W}}_{::k^{opt}}$ and $\mathcal{H}^{reg}_{::k^{opt}}$ are the robust solutions for the low-rank factor matrices.

We provide a sample Silhouette score and relative error plot produced by *NMFk* for two factorizations, to demonstrate the selection of k, in Figure 1. The presented *NMFk* framework estimates the latent feature count based on two criteria, namely a high minimum Silhouette score, and a low relative reconstruction error, which corresponds to a stable NMF solution. The number of features with lower minimum Silhouette scores correspond to overlapping clusters or scattered clusters. On the other hand, the relative reconstruction error decreases monotonically with the number of latent features. This decrease is more prominent up to the estimated number of topics followed by a reduced change in the error. As observed in Figure 1, with the further increase in the number of latent features past the estimated k, there is a sudden decline in the Silhouette score due to the over-fitting phenomenon as the model tends to fit noise.

3.3 Hierarchical Non-Negative Matrix Factorization

The NMF and Hierarchical NMF [31, 43] strategies have been used successfully for document clustering [20, 72], and topic modeling [30, 62, 70]. Here we use *NMFk* to compute clusters of malware specimens by applying it in a hierarchical manner, where successive node expansions focus on the subset of $\mathbf{X}$ obtained from the parent cluster. Here the clusters are determined using the columns of $\mathbf{W}$ via *W-clustering* that cluster the specimens' coordinates in the reduced space (i.e. the rows of the matrix $\mathbf{W}$) [68], a topic we address in Section 3.4. When going deeper in the graph towards the leaves, we investigate different characteristics of the specimens in the same group, and achieve better separability of the malware specimens.

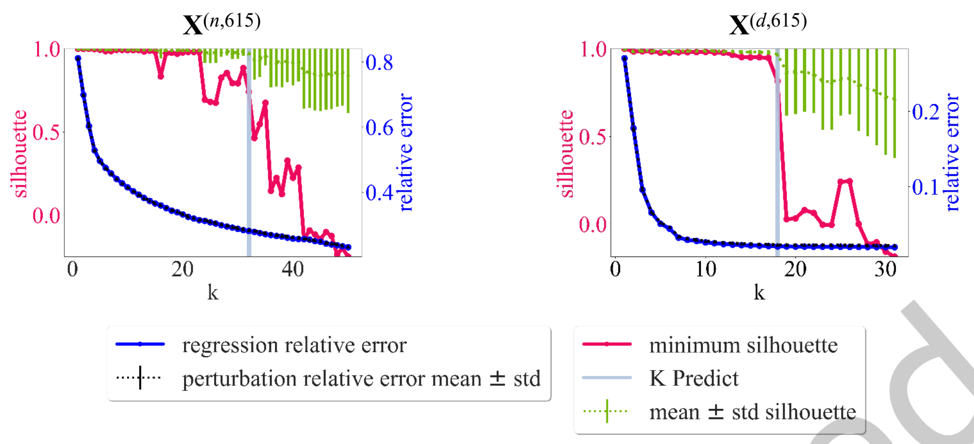

Fig. 1. Sample Silhouette and relative error graphs obtained from NMFk is shown for the matrices $\mathbf{X}^{(n,615)}$ and $\mathbf{X}^{(d,615)}$ which are formed using 1,000 malware specimens from 10 families. $\mathbf{X}^{(d,615)}$ consist of samples extracted from a single cluster after the the first NMFk procedure on $\mathbf{X}^{(n,615)}$.

Let us consider a simple example from hierarchical document clustering. We assume three well-curated clusters in a text corpus of news articles about sports, technology, and the economy. If we cluster these documents with *NMFk* and *W-clustering* we can obtain three *"super"* clusters for sports, technology, and the economy. We can further divide the cluster containing sport articles into sub-topics, such as soccer, football, tennis, skiing etc. by applying additional iterations of *NMFk*. In our analysis, we choose to select back the specimens corresponding to each one of the super clusters and apply *NMFk* again. This is the idea behind the hierarchical approach, and consequently a hierarchical approach is used in the *HNMFk Classifier* to further separate more heterogeneous clusters based on the known (or labeled) malware instances.

In our semi-supervised setting, the experimental setup contains data with labeled (known) and unlabeled (unknown) malware specimens. This allows us to choose a scoring function, not based on information gain (such as normalized discounted cumulative gain from information retrieval [39]) [43] or a fixed threshold using the number of specimens in the cluster [31] to determine which node to take further. Instead, we use a *cluster uniformity score* that measures the stability of the cluster, based on the known specimens in the cluster, as the node expansion criteria for a cluster. We will further explain how we calculate the cluster uniformity score in Section 3.4. In general, the application of *NMFk* to semi-supervised data will place into each of the final clusters both labeled and unlabeled malware specimens. This allows us to continue to build the hierarchical graph until the further expansion of a particular node is stopped if no unlabeled or labeled samples are present in this node, or if the cluster uniformity score calculated based on the known samples passes the provided threshold.

We provide an example visualization of the latent factors obtained from *NMFk* with a hierarchical setting in Figure 2. Here, we apply dimensionality reduction using *t-SNE* [67] to each latent factor **W** to plot the clusters. Each point in the embedding of **W** is colored based on the family to which the specimen belongs. Here the clusters are expanded until all the samples in the cluster belong to a single class. The *t-SNE* visualization show how the hierarchical clusters of malware families are formed, and how the clusters become more homogeneous as we perform additional applications of *NMFk*.

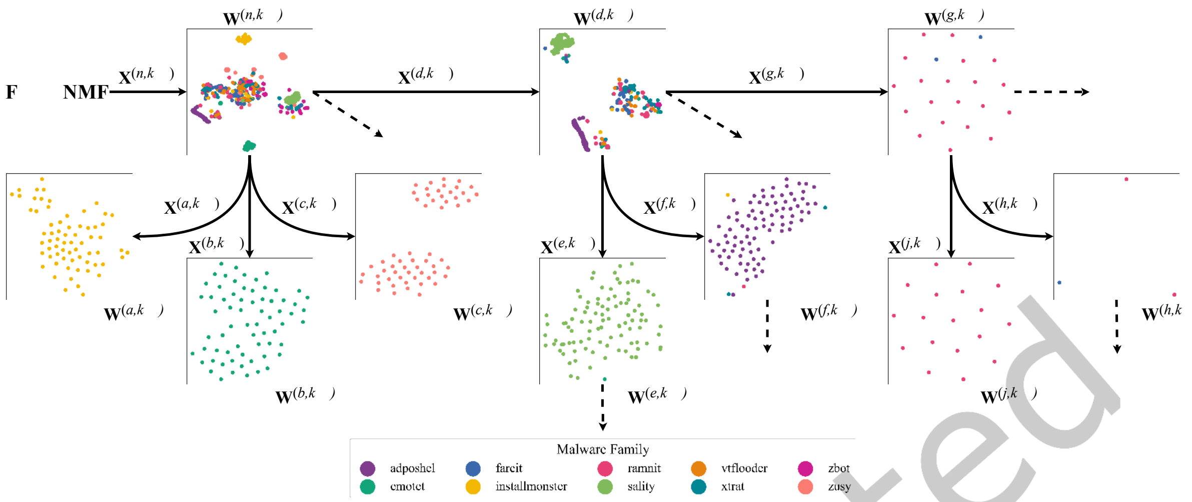

Fig. 2. The path of the hierarchical graph formed by the NMFk is shown using 1,000 malware specimens containing a total of 10 malware families. After each factorization, the clustering is visualized by reducing the dimensions of **W** using t-SNE. Dashed arrows are used to indicate the existence of an another sub-tree from the node. Since we are obtaining k_{opt_i} subsets of specimens from current **X** at each stage, $n > a \geq b \geq c \geq d > e \geq f \geq g > j \geq h$.

3.4 HNMFk Classifier

In this section, we describe how our *NMFk* based hierarchical bulk classifier works. Our model recursively analyzes the known and unknown specimens, factorizing only the subset of data from the previous cluster at each iteration.

The hyper-parameters of our model are the hyper-parameters needed for *NMFk*, and the cluster uniformity threshold t. The user specifies the maximum number of iterations for NMF, number of perturbations, the error rate, and the range to search for the k heuristic. When performing classification, *HNMFk Classifier* is provided with the data matrix $\mathbf{X} \in \mathbb{R}_{+}^{n \times m}$, where n is the number of malware samples and m is the number of features, which includes both the known and unknown specimens that we want to perform inference on. We also provide a vector $\mathbf{y}$ containing the labels for each specimen. The ith sample, where $1 \geq i \geq n$, has the family label $\mathbf{y}_i \in \{-1, 1, 2, \ldots, C\}$ for a dataset with C classes. Notice that the unknown specimens are labeled with -1.

Our algorithm proceeds with the first factorization, given $\mathbf{X}$, $\mathbf{y}$, the specified *NMFk* hyper-parameters, and cluster uniformity threshold t as input. After *NMFk* identifies the number of clusters k^{opt}, we obtain the latent factors $\mathbf{W} \in \mathbb{R}_{+}^{n \times k^{opt}}$ and $\mathbf{H} \in \mathbb{R}_{+}^{k^{opt} \times m}$. *HNMFk Classifier* uses $\mathbf{W}$ latent factor to perform clustering, which we call *W-clustering*. Here each n sample is assigned to one of k^{opt} clusters by taking the maximum value along the second axis:

$$\text{cluster}(i) = \underset{0 \leq j \leq k^{opt}}{\arg\max}\,(\mathbf{W}_{ij}) \tag{1}$$

where cluster(i) returns the cluster assignment of a given sample i. If a cluster c, where $c \in \{1, 2, \ldots, k^{opt}\}$, does not contain any known samples, all the unknown specimens in the cluster c are predicted abstaining. If a cluster c only has known specimens, we do not proceed with the samples in that cluster further, as there are no more unknown specimens to label. On the other hand, if a cluster c has a mix of known and unknown samples, we calculate the uniformity of the cluster based on the known specimens. Our cluster uniformity score is defined by the fraction of the most dominant class present in the cluster c:

$$U^c = \frac{|\max(c^{known})|}{|c^{known}|} \tag{2}$$

where U^c is the cluster uniformity score for the cluster c, $|c^{known}|$ is the number of known samples in the cluster, and the numerator is the number of samples that belongs to the most dominant known class in c. U^c specifies how uniform the given cluster c is based on the labeled data.

If the cluster uniformity score U^c is more than the threshold t, then we proceed to assign unknown specimens family labels in a semi-supervised fashion. That is, all the unknown samples are predicted to be the most dominant class in the cluster based on the known specimens ($\max(c^{known})$). If, however, the cluster uniformity score is less than the threshold t for a given cluster c, we form a new $\mathbf{X}' \in \mathbb{R}_{+}^{|c| \times m}$ that only contains the malware specimens present in that cluster (both known and unknown). Finally, $\mathbf{X}'$ is factorized again with *NMFk*. In the proceeding *NMFk*, k search range selected to be $[1, k^{opt}]$ with the step-size of 1. The above procedure is repeated until all the unknown samples are classified. In this setting, our leaf nodes in the hierarchical graph are the positions where at least one of the following exit conditions are met: no known specimens in the cluster (abstaining prediction), no unknown specimens are in the cluster (nothing to classify), or $U^c \geq t$ is true and we classify all samples in the cluster in a semi-supervised manner. The aforementioned procedure is summarized in Algorithm 2 and Figure 3.

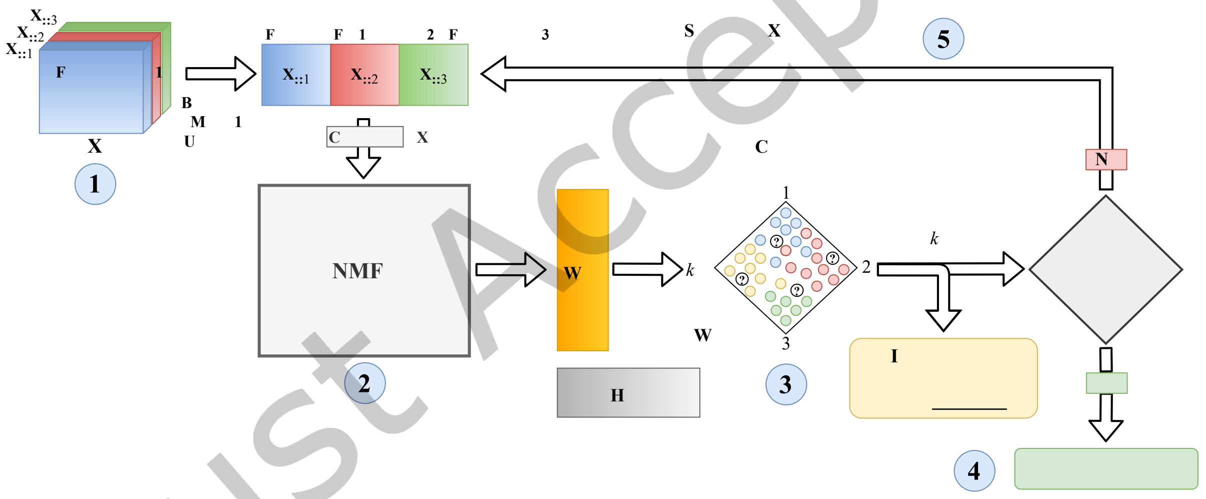

Fig. 3. Overview of the HNMFk Classifier framework. NMFk is wrapped around an hierarchical (or recursive) semi-supervised architecture. Begin with the initial data **X** (1). Use NMFk to estimate the number of clusters and obtain the latent factor **W** (2). Extract the clusters via argmax along the second axis of **W** (3). For each cluster, perform abstaining prediction if no known samples are present in the cluster, or predict the unknown specimens in a semi-supervised manner if the cluster uniformity score is satisfied (4). Form the new matrices **X** with the specimens from the clusters that does not meet the cluster uniformity threshold (5). For each new **X**, apply NMFk again (2).

In summary, looking at Figure 3 we can conclude that *HNMFk Classifier* is a wrapper to the *NMFk* algorithm, which exploits *NMFk*'s ability to estimate the number of latent components, and performs factorization recursively to create a hierarchical graph where the semi-supervised classification is done at each leaf node. When our model

Algorithm 2 HNMFk Classifier($\mathbf{X}$, $\mathbf{y}$, k^{min}, k^{max}, r, t) - Semi-supervised Hierarchical Classifier

```
known_samples= argwhere(y != -1)
unknown_samples = argwhere(y == -1)
$\mathbf{W}$, $\mathbf{H}$, $k^{opt}$ = NMFk($\mathbf{X}$, $k^{min}$, $k^{max}$, $r$)
clusters = argmax($\mathbf{W}$, axis=1)
for cluster in clusters do                                    ▷ iterate over $k^{opt}$ clusters
    known_samples_c = intersect(known_samples, cluster)
    unknown_samples_c = intersect(unknown_samples, cluster)
    if len(known_samples_c) == 0 then             ▷ no unknown samples to make prediction
        continue
    end if
    if len(unknown_samples_c) == 0 then                               ▷ abstaining prediction
        continue
    end if
    class_counts = count(known_samples_c)
    cluster_uniformity = max(class_counts) / sum(class_counts)
    if cluster_uniformity < t then
        X_new = X[cluster]                          ▷ subset in X, samples in the cluster
        y_new = y[cluster]                        ▷ labels for the samples in the cluster
        $k^{max}$ = min($k^{opt}$+1, min(X.shape))
        y[cluster] = HNMFk_Classifier(X_new, y_new, $k^{min}$, $k^{max}$, r, t)
    else
        classify_label = max(class_counts)       ▷ dominant known class in the cluster
        y[unknown_samples_c] = classify_label
    end if
end for
return y
```

finishes classification, any unknown samples that are left with the label -1 are said to be abstaining predictions, i.e. the model does not know their classes or rejects to make a prediction.

We also provide a toy example illustrating how *HNMFk Classifier* works in Figure 4. In this figure, we have a matrix $\mathbf{X} \in \mathbb{R}^{9 \times 3}$ (9 malware samples with 3 features). After factorizing $\mathbf{X}$ with *NMFk*, we get the latent factors $\mathbf{W} \in \mathbb{R}_{+}^{9\times k^{opt}}$ and $\mathbf{H} \in \mathbb{R}_{+}^{k^{opt}\times 3}$, with the estimated number of clusters $k^{opt} = 4$. Samples 5 and 6 are assigned to cluster 2. Sample 6, an unknown sample, is classified as family a. Cluster 3 contains only 2 unknown samples. Therefore, we classify samples 2 and 7 as abstaining. Cluster 1 contains the sample 1 (family a), 3 (family b), and 8 (unknown). Because this cluster has samples belonging to two different families (assuming that our cluster uniformity threshold is $t = 1$, i.e. threshold is met only when all the known samples in the cluster belongs to a single class), we create a new subset with these samples, such that $\mathbf{X}' \in \mathbb{R}_{+}^{3\times 3}$. We apply *NMFk* again on $\mathbf{X}'$, which estimates $k^{opt} = 2$, and sample 8 is classified as family b. When all samples are predicted, the computation is complete.

4 DATASET

Collection of malware data has challenges such as copyright issues, labeling difficulty, and security precautions. Therefore, compared to other ML fields with abundant data (such as text and images), the malware identification

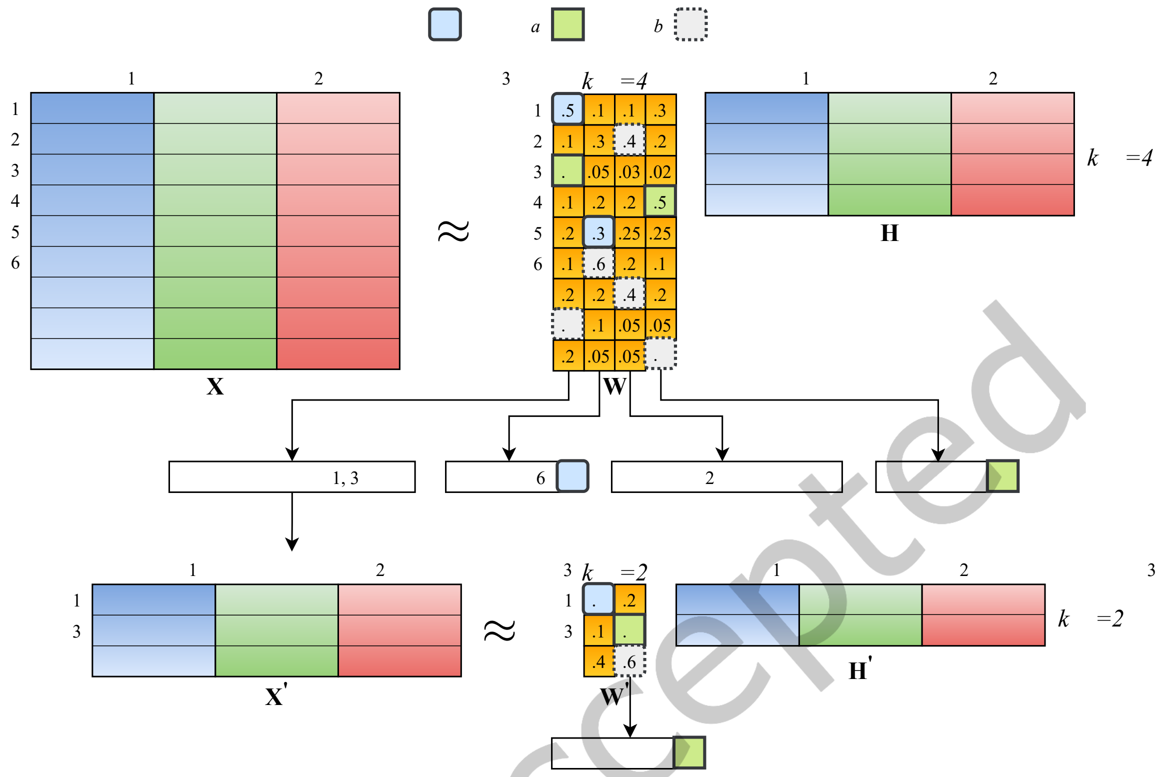

Fig. 4. A toy demonstration of how HNMFk Classifier operates in a hierarchical fashion, and how the semi-supervised classification of the unknown malware specimens is performed via the clustering on the latent W matrices using the known samples.

community has lacked a benchmark dataset sufficient to enable reproducibility and comparison of new methods. To address this issue, Anderson et al. released the EMBER-2018 dataset [8], which we use in our experiments. Since its release, EMBER-2018 has become a popular benchmark dataset for ML-based malware analysis methods.

EMBER-2018 is a collection of PE header and meta-data information extracted from 1.1 million benign and malicious Microsoft Windows Portable executable binaries, out of which 800,000 have labels. The family labels in the dataset are obtained using AVClass. Therefore, classes are *weakly labeled* as AVClass contains inaccuracies in family labeling [76]. AVClass does not filter out all generic family names, it can be inconsistent in its use of aliases for malware families, and errors in any antivirus signatures can effect AVClass' accuracy. Despite the imperfect labeling, AVClass is currently the best available option for obtaining a large quantity of malware family labels.

Throughout our analysis, we only use the malware instances for which AVClass could determine a family label. The final dataset includes a default train and test split, where the training set consists of over 289,000 specimens from 2,730 malware families, and the test portion of the dataset includes around 99,000 samples from 916 malware families. The details of the dataset sample and family statistics are shown in Table 3. In this paper, we refer to the training set as *known* data, and testing set as *unknown* data in the context of semi-supervised learning and bulk classification (i.e. we use the *known* data as a reference to label the *unknown* data).

One advantage of using the EMBER-2018 dataset is that the distribution of the family classes resembles real-world cases. The *known* portion of the dataset contains malware families that do not exist in the *unknown*

Table 3. EMBER-2018 dataset default train and test set split and malware family and sample counts are displayed. Novel families for the known (or train) set are the families that only exist in the training set. The novel families for unknown (or test) set are the families that only exist in the test set (i.e. we do not see these families during inference, or we do not have known specimens for reference). Min Family and Max Family columns show the minimum and maximum number of samples exist for a family in the dataset. For instance, there are malware families with single sample in both known and unknown sets. Samples/Family column shows the average number of samples per family. We used all the malware instances with a family label from EMBER-2018, which contains the rare and novel families, making the classification task complex.

Set	**Families**	**Samples**	**Novel Families**	**Novel Samples**	**Min Family**	**Max Family**	**Samples/Family**
Known (Train)	2,730	289,026	1,982	11,157	1	16,689	105.87
Unknown (Test)	916	99,216	168	363	1	19,260	315.53

portion of the data. Similarly, the *unknown* set contains novel malware families, or the malware families that do not exist in the *known* set. This is also shown in Table 3. 1,982 of the malware families, making over 11 thousand samples, are not seen again in the *unknown* set. There are 168 novel families, forming 363 samples, that we do not have any reference of in the *known* set. At the same time, malware family classes in EMBER-2018 are extremely imbalanced. Figure 5 shows the distribution of the malware families for both the *known* and *unknown* set. For instance, there are malware families that consist of single samples, including the specimens from the novel families (which can also be seen at the right side of Figure 5 with red-dashed line). In fact, the majority of the malware families in the dataset consist of less than 10 samples. We next proceed to the pre-processing of the features to remove the outliers.

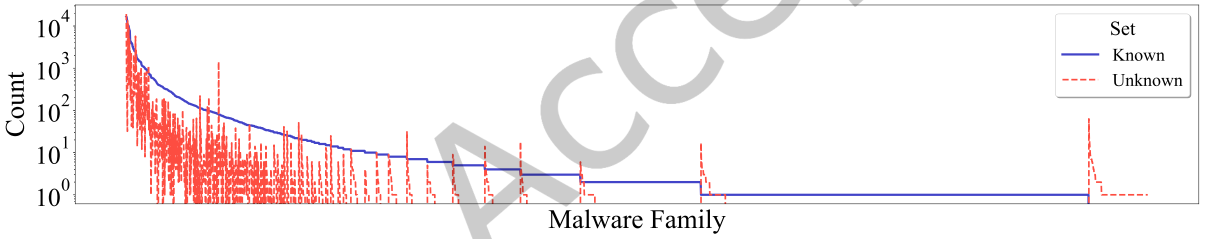

Fig. 5. Distribution of the malware families in EMBER-2018 dataset. Count of family classes are shown in log scale for both the known and unknown set. Both the known and unknown sets has an extremely imbalanced classes of malware families, and the unknown set of contains set of novel malware families.

4.1 Pre-processing

During our experiments, we represent each file in the dataset as a collection of features from both general file meta-data as well as PE header information. Each of the features are concatenated horizontally to form the final features matrix. This is equivalent to forming an 11 dimensional tensor, with the dimensions *Samples* × *Feature 1* × *Feature 2* × ... × *Feature 10*, and taking the mode-1 unfolding of the tensor. Specifically, we use the following features:

(1) *byte histogram*: a vector of size 256 where each entry represents the number of times a certain byte occurs in the file.
(2) *byte entropy*: normalized joint distribution of entropy and byte values.
(3) *print table distribution*: distribution of characters obtained from printable strings with minimum of 5 consecutive printable characters in the binary.
(4) *strings entropy*: measure of randomness of printable strings present in the malware.

(5) *number of strings*: number of printable strings.
(6) *file size*: size of the binary in bytes.
(7) *number of exports*: number of functions exported by the malware.
(8) *number of imports*: number of functions imported by the malware.
(9) *code size*: size of *.text* or code section of the PE header in bytes.
(10) *number of sections*: number of sections present in PE header.

Our dataset consists of heterogeneous features containing outlier values. Since NMF is susceptible to outliers (extremely large or small values in the columns of initial data), see for example [74], we normalize the features used in our analysis. This normalization prevents the larger values in the columns of the initial data to bias/skew the NMF optimization procedure by favorizing some of the columns in **X**, see details in Ref.[37]. Note that the case of outliers affecting NMF optimization is distinct from the characteristics makeup of a novel malware family: After the normalization, the novelty of the malware belonging to unknown (or never seen before) family reflects on the shape of its latent signature (the columns of matrix **W**). A possible classification of malware families based on their latent signatures will be discussed elsewhere.

In our normalization, Z-scores are used to remap the outliers that are more than or less than 3 standard deviations away from the mean. These outliers are mapped to the point that is exactly 3 standard deviations away from the mean. In Figure 6, we show the histogram of feature values for pre- and post-processing. The normalization was most prominent among the features *byte histogram*, *byte entropy*, and *print table distribution*. Finally, we scale the values to be between 0 and 1.

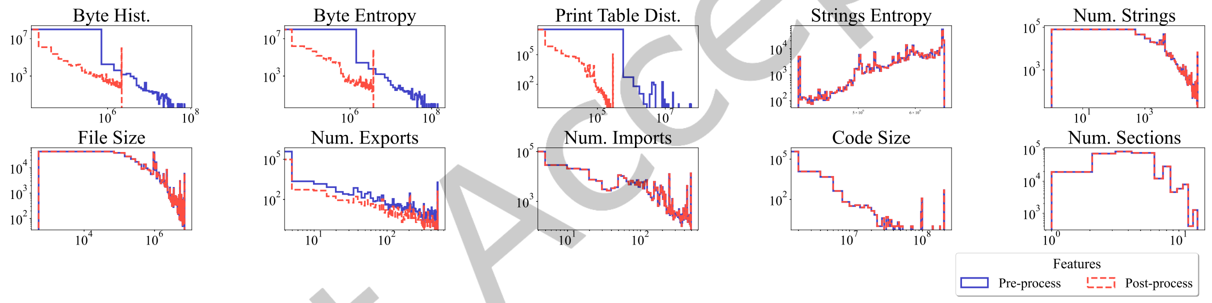

Fig. 6. Static malware analysis based features from PE header files and malware meta-data used in the analysis shown before and after the mapping of the outliers, defined by $Z = 3$ statistical score, for both training (known specimens) and test sets (unknown specimens).

4.2 Preparation of the Experiments

We conduct our experiments using two different dataset setups. With the first setup, we use a subset of data utilizing only the top populous malware families to perform performance analysis of our method under different conditions in Section 5.1. This setup is also used in our ablation studies in Section 5.3. We use a smaller subset of the data to reduce the computation time of our experiments and to enable testing of our method under number of different settings. Although this setup allows us to gain insights into how our method works, it does not yield results that can generalize to real-world. Therefore, in Section 5.2 we test our method under realistic conditions and compare to other baseline models.

In our small dataset setup, we chose the 10 most populous malware families in the entire dataset (adposhel, emotet, fareit, installmonster, ramnit, sality, vtflooder, xtrat, zbot, zusy). We then randomly sample the dataset to extract 1,000 specimens for each family without replacement, forming a small subset of the dataset with 10,000 samples. We form 10 of these random subsets, and apply our experiments on each of the 10 subsets, to see if our results are statistically significant. In our large scale analysis, we used all of the malware families present in the EMBER-2018 dataset, and use the default split present in EMBER-2018 to separate the known and unknown specimens.

4.3 System Configuration

We ran the experiments on a High Performance Computing (HPC) cluster named Dracarys, located at the Los Alamos National Laboratory (LANL). Dracarys uses the Intel(R) Xeon(R) Platinum 8280M processor, which is a cascade lake architecture operating at a clock speed of 2.70GHz. There are 28 physical CPU cores which are multi-threaded to 56 threads providing 112 virtual processors, and total physical RAM of 2.71 TeraBytes (TBs). The system also comprises 3 NVIDIA Quadro RTX8000 GPUs with VRAM memory of 48 GigaBytes (GBs) each.

5 EXPERIMENTS

We perform experiments targeting the following tasks: analyzing the performance of our model with different hyper-parameters and, as the amount of known malware decreases, and testing our method under realistic conditions. We compare our results to those obtained by the baseline models, taking advantage of the abstaining prediction ability to detect novel malware, and using ablation studies to justify the need for the parts of our model.

5.1 Methodology Performance Analysis

In this section we look at the performance of our method for different cluster uniformity thresholds, unknown malware fractions, and *NMFk* hyper-parameter selections. Similar to prior work, we use a small subset of the dataset (an unrealistic data setup), as described in Section 4.2, during our analysis in this section. Each experiment is run 10 times on different random subsets of the dataset, to verify if the results are statistically significant using hypothesis testing. To this end, we report our results with a 95% confidence interval (CI) for each experiment.

5.1.1 Cluster Uniformity Threshold. We use a threshold value t, which measures how many labeled (known) specimens are needed to claim that all unknown specimens in this cluster are uniform, that is, from the same labeled malware family. This threshold allows us to determine whether to proceed further with clustering of the current data in the node with additional applications of *NMFk*. The left side of Figure 7 shows the percent of abstaining predictions, execution time, and the maximum graph depth (maximum number of edges between the root and a leaf node) as the cluster uniformity threshold t is changed. As t increases, the percent of abstaining predictions rises, since the solution needs increasingly cleaner clusters. This reduces the number of specimens that we can classify with high certainty, and results in a higher number of abstaining predictions. The maximum graph depth also increases, alongside the higher execution time, since achieving cleaner clusters requires an increased number of separations. We show how the F1 score changes for each malware family in Figure 8. As the cluster threshold increases, the performance of the model improves for each malware family, and the results become more certain, as indicated by the narrowing confidence interval. Although the computation time increases, a higher threshold yields better inference results. Therefore, during the experiments in Section 5.2 we set the threshold to be $t = 1$.

5.1.2 Unknown Malware Fraction. The process of labeling malware is expensive [58]; therefore, semi-supervised learning can help with obtaining good performance results when using a low quantity of labeled data. We investigate this by looking at how our model performs as the unknown malware fraction increases. Figure 9 displays the average F1 score for each malware family as the unknown malware fraction rises. Since our model can perform abstaining predictions, as the unknown malware fraction increases, the performance of the model remains relatively stable. A lesser number of known malware samples means that our model to have a

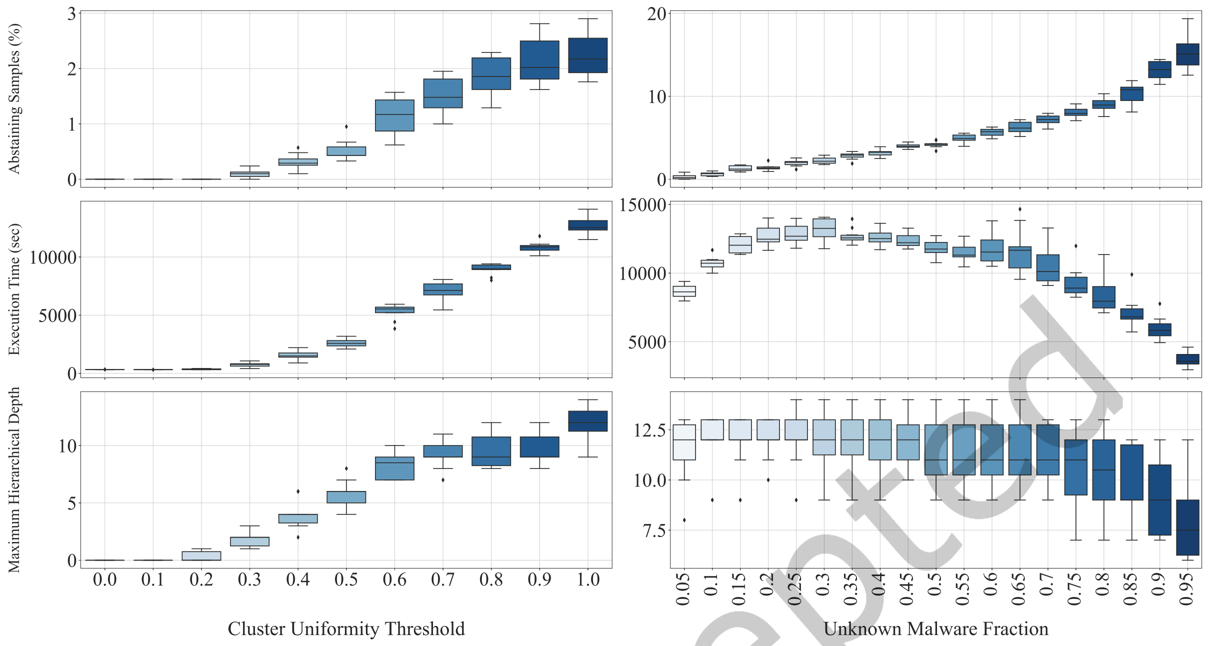

Fig. 7. HNMFk Classifier's performance for abstaining prediction, execution time, and the maximum depth is shown as the cluster uniformity and the unknown malware fraction changes.

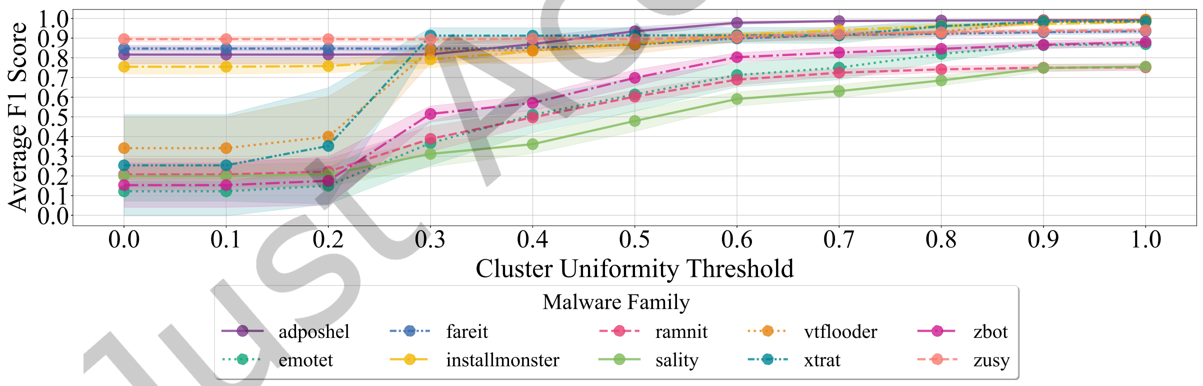

Fig. 8. The performance of HNMFk Classifier is measured with the F1 score as the cluster uniformity threshold is changed. Each experiment is performed on 10 different random subset of the EMBER-2018 dataset, average is plotted with the 95% confidence interval.

lesser number of references that can be used to classify the unknown samples. This results in higher number of abstaining predictions which in return helps with maintaining the performance (this can be seen at the right top of Figure 7). In Figure 9, we can also see that two malware families, Sality and Ramnit, yield lower F1 scores in comparison to the other families. Possible reasons for diminished performance on Sality and Ramnit include the fact that they are both *"file infectors"* (a category of malware which copies its code into other executables). It may be more difficult to classify this type of malware using the selected features, since some of the original PE metadata/file contents may not be changed when a file is infected.

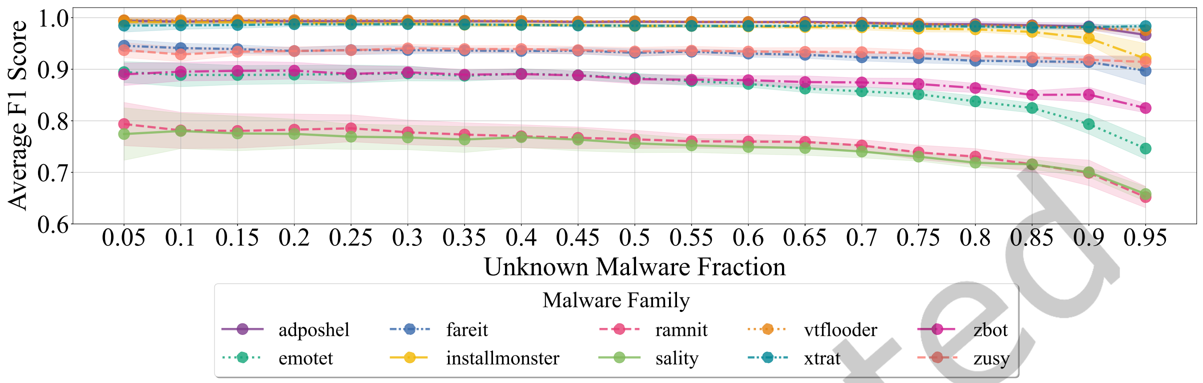

Fig. 9. The performance of the HNMFk Classsifer, measured with F1 score, remains relatively stable for each malware family as the unknown malware fraction increases (or the number of known samples decreases). Each experiment is run on 10 random subset of the dataset.

The average F1 scores obtained by the *HNMFk Classifier* with the changing unknown specimen fraction are also compared to the vanilla baseline models in Figure 10. Here, the unknown malware fraction point where the *HNMFk Classifier* begins to outperform a baseline model is shown with a vertical line. We use the supervised baseline models *XGBoost* and *LightGBM*, and a semi-supervised model *LightGBM+SelfTrain*. These traditional ML models do not have the ability to perform abstaining predictions. Therefore, they rely on an abundance of labeled data to perform well during testing. The *HNMFk Classifier* surpasses the average F1 score of *LightGBM+SelfTrain* at 0.64 unknown malware fraction. *XGBoost* is outperformed at unknown malware fraction 0.94, and *LightGBM* at 0.97. We also note that these models continue to perform relatively well as the known malware fraction drops because we are using a small and balanced subset of the dataset which contains the most populous malware families, making the problem easier. We will be further analyzing the performance of the baseline models and our approach with a realistic dataset setup in Section 5.2. The experiments under real-world like setup will reveal that the performance difference between the baseline models and our method is even greater.

5.1.3 NMFk Hyper-parameter Analysis. In addition to the cluster uniformity threshold hyper-parameter of the *HNMFk Classifier*, we also provide our model with the hyper-parameters of *NMFk*. In Figures 11 and 12 we show that changes in the number of perturbations and NMF iterations do not have a large effect on the performance of our method. Figure 13 displays the change in F1 score as the maximum k is increased for the k search of first *NMFk*. In this experiment, we choose the k step-size of 1, and begin searching at $k = 1$. The performance of the model continues to increase as the predicted k is approached. After the estimated k^{opt} is reached, the F1 score does not change, since we will always choose the same k^{opt} in the first *NMFk*. These experiments indicate that we need to choose the initial k search range to be large enough to obtain a good initial clustering.

5.2 Malware Family Classification Under Realistic Conditions

Now that we have gained understanding into how our method performs with different hyper-parameters and settings, we will next use the more realistic data setup to show how our approach fares far better under real-world constraints. When ML-based malware defense and analysis solutions are used outside the research environment,

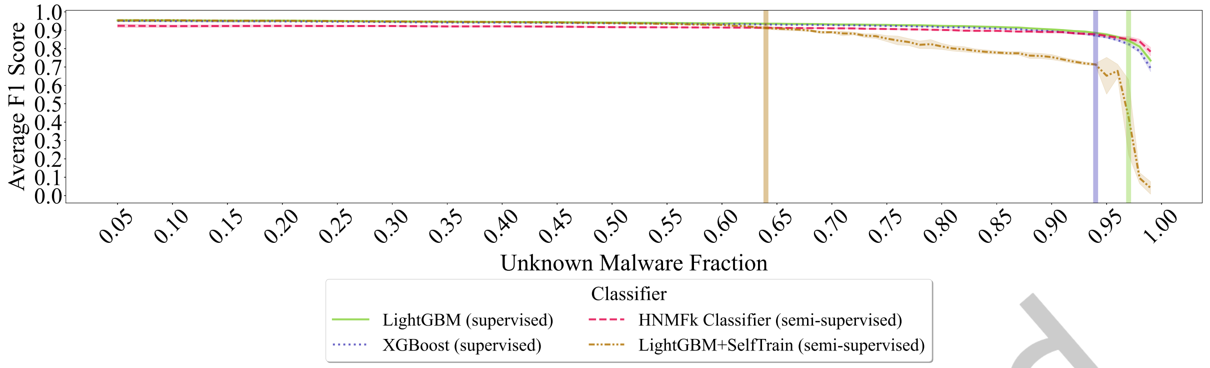

Fig. 10. Average F1 score when classifying 10 malware families is compared to other baseline models as the fraction of unknown malware increases. Each experiment is run on 10 random subset of the dataset.

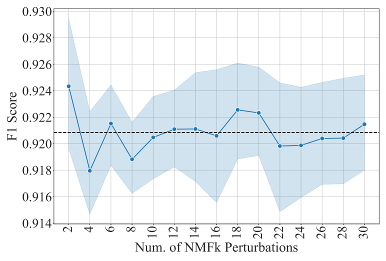

Fig. 11. Change in performance is measured using F1 score as the number of NMFk perturbations increased. It can be seen the effect to the overall performance as this hyper-parameter is changed is low, with average average F1 score of .92 and confidence interval .001. The difference between the highest and lowest point is .032.

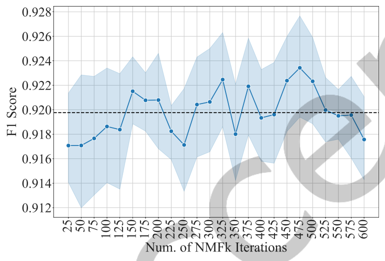

Fig. 12. Change in performance is measured using F1 score as the number of NMFk iterations increased. It can be seen the effect to the overall performance as this hyper-parameter is changed is low, with average average F1 score of .91 and confidence interval .0008. The difference between the highest and lowest point is .031.

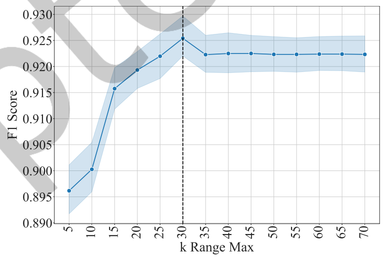

Fig. 13. Change in performance is measured using F1 score as the maximum k for of NMFk k search is increased. Here, k range is 1 through the maximum k value, with the step size of 1. It can be seen as the maximum k increases, the performance of the model improves. After the estimated k, increasing the value further does not change the performance.

they often encounter extreme class imbalance. At the same time, analysts do not have access to all possible malware samples, and threat actors continuously develop new pieces of malware. Therefore, ML-based systems are exposed to malware that has never been seen before. To this end, we analyze the performance of our method under a real-world like setting by exposing our model to prominent, rare, and novel malware families. In this section, we utilize all the malware families present in the EMBER-2018 dataset to conduct our experiment, as described in Section 4.2. The performance of *HNMFk Classifier* is compared to the supervised baseline models *LightGBM*, *XGBoost*, and *MLP*. We also create strong semi-supervised versions of *LightGBM* and *XGBoost* by wrapping them with *SelfTrain*. During the hyper-parameter tuning of *LightGBM* and *XGBoost*, we use the Python package *Optuna* to get the hyper-parameter suggestions for each trial [2], and for the construction of an optimal neural net-based classifier, *MLP*, we employed a *HyperBand Tuner* as an accelerated tuning algorithm [45]. The hyper-parameters of *LightGBM* was tuned using a stratified 20% subset of the training set over 65 trials and 3-fold stratified cross-validation. We used a stratified subset of the dataset because using the entire dataset for this model resulted in each trial taking approximately 2 days during tuning (it would have taken approximately 100 days to complete 50 trials for tuning). We used the objective *multiclass* with a 500 maximum number of iterations, and *gbdt* boosting type. 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). For *LightGBM*, we have also tried the recommended hyper-parameters from the EMBER-2018 dataset [8], which did not yield better results when compared to our best trained model.

XGBoost was tuned using the entire dataset over 25 trials with stratified 3 fold cross-validation. We used maximum boosting rounds of 500 with the multi-class softmax objective function. The following hyper-parameters were tuned: *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).

The *HyperBand* framework has been widely used in the deep learning community for estimating the optimal parameters in a short amount of time. HyperBand is a variation of random search with explore-exploit theory to estimate best configurations within a given allocated time. The hyper-parameters utilized for model selection of the *MLP* were the number of depths of the neural network (1-10), number of nodes on each layer (1024-16000), optimization algorithm (SGD, Adam, RmsProp), and the learning rate (1e-4, 1e-1). We employed early stopping criteria on validation loss to avoid over-fitting.

Since our method's performance does not change dramatically with the change in hyper-parameters as shown in Section 5.1.3, we choose the hyper-parameters without tuning with 20 perturbations, 500 number of iterations, and k-range to be 1 through 100 with the step-size of 1 for the first iteration. We did verify, by inspecting the plot of the initial *NMFk* (similar to the Figure 1), that the estimated number of components was less than 100. If it had been close to 100, we would have re-started our experiment with a higher range. Finally, we chose the cluster uniformity threshold t to be 1, i.e. each cluster should have a single known class to be able to perform semi-supervised classification.

Table 4 compares our method to the baseline models. The *HNMFk Classifier*, a semi-supervised solution, outperforms all of the state-of-the-art models, which we used as baselines, with an F1 score of 0.80. Our approach outperforms the supervised methods, with the potential benefit of better generalizability and the need for less labeled data, due to the semi-supervised setting. We also surpass the strong semi-supervised version of *XGBoost* with *SelfTrain*. Notice that these baseline models were used to report benchmarks by prior studies. However, our experiment reveals the performance of these models under realistic conditions.

Table 4. HNMFk Classifier is compared against the state-of-the-art supervised classifiers. HNMFk Classifier, a semi-supervised method, surpasses the previous state-of-the-art models, which are supervised, in malware family classification. **Weighted** F1, Precision, and Recall scores are provided for multi-class classification with imbalanced data. The F1 scores of HNMFk Classifier and HNMF2 Classifier does not include the abstaining predictions (score includes the specimens where the prediction was not rejected)

Model	F1	Precision	Recall	Tune Time	Train&Predict Time
HNMFk Classifier (semi-supervised)	**0.80**	**0.85**	**0.77**	5.77 days	7.91 days
HNMF2 Classifier (semi-supervised, ablation study)	0.77	0.82	0.74	5.77 days	2.83 days
XGBoost+SelfTrain (semi-supervised)	0.76	0.78	0.73	2.06 days	4.72 hours
XGBoost (supervised)	0.74	0.77	0.72	2.06 days	2.93 hours
LightGBM (supervised, tuned on stratified subset)	0.65	0.74	0.64	11.09 days	3.02 hours
MLP (supervised)	0.72	0.76	0.71	1.02 days	30 minutes
LightGBM+SelfTrain (semi-supervised)	0.64	0.69	0.61	11.09 days	9.44 hours

Additionally, our method utilizes abstaining predictions (rejection to make a prediction), which other baseline models do not perform. We provide the metrics for the abstaining predictions in Table 5. The models that do not perform abstaining predictions always predict the novel specimens incorrectly since these samples belongs to a new class. The proposed ability to predict novel samples as *"other"* may still require the model to have seen the given specimen in the *"other"* class, which is not as effective as rejecting to make a prediction, which incorporates uncertainty in the model. In addition, as pointed out by Loi et al. [47], predicting specimens as *"other"* class often results in false predictions due to supervised models' common inability to learn patterns from a small number of samples. Our method novel ability to reject making a prediction yields promising results in identification of novel malware. Interestingly, around 22% of the malware which we saw in the known set were also predicted as abstaining by the *HNMFk Classifier*. This 22% we referred as *false-abstaining*, since the specimens here belongs to classes that we had labels for. Importantly, around 42% of the novel malware (i.e. the malware which we did not see in the known set), are classified as abstaining. This 42% is referred as *true-abstaining* since our model did not have a reference label for these specimens in the known set. We also note that both true and false abstaining predictions would be caused by signatures or patterns extracted by *NMFk* being distinct from the labeled samples. Hence, it is possible that a detailed investigation and utilization of latent signatures can help to reveal characteristics that differ given specimen from the known samples (similar to the prior work in latent mutational cancer signatures [5]) and result in improvement of the abstaining predictions, as also shown by the follow-up work [23].

In Table 5, for completeness, we also provide F1 scores for each baseline that is calculated only of the specimens that HNMFk Classifier did make a predictions (i.e. it did not abstain). Notice that the F1 scores of our baselines increase, even surpass our model in some cases, when the rejection to make predictions is not included in the score calculations. This result points out that the abstained samples are hard to correctly classify since our baselines yield lower scores when they are included (see the scored reported in Table 4). While the baselines falsely predicted the families for the harder specimens, HNMFk Classifier rejected to make a prediction and managed to maintain higher performance.

Table 5. HNMFk Classifier is compared against the state-of-the-art supervised classifiers. The ability of the HNMFk to discover novel families is shown. **F1 - (Non-reject)** column shows the F1 scores for the specimens that HNMFk Classifier did make a prediction on. Not applicable (NA) used at the cells where the case does not apply to the given model. **Abstaining Seen** refers to false-abstaining predictions, samples that belong to known classes that were seen in the training set. Differently, **Abstaining Novel** shows the true-abstaining predictions, where the specimen belongs to a class that were not seen before.

Model	**Abstaining Seen (%)**	**Abstaining Novel (%)**	**F1 - (Non-reject)**
HNMFk Classifier (semi-supervised)	22.06	**42.70**	0.80
HNMF2 Classifier (semi-supervised, ablation study)	**16.96**	34.16	0.77
XGBoost+SelfTrain (semi-supervised)	NA	NA	0.81
XGBoost (supervised)	NA	NA	0.80
LightGBM (supervised, tuned on stratified subset)	NA	NA	0.74
MLP (supervised)	NA	NA	0.79
LightGBM+SelfTrain (semi-supervised)	NA	NA	0.70

We also apply our ablation study, where the number of cluster selection heuristic is turned off and rank-two factorization is used (i.e. $k = 2$ at each node). In table 4, we can see that the *HNMF2 Classifier* does perform better than our baseline models. However, the *HNMFk Classifier* outperforms this method, which points out that carefully choosing the number of clusters improves the separability and the overall performance during prediction. *HNMF2 Classifier* also reduces the percent of abstaining predictions, including the reduced percent of abstaining predictions on novel malware. We show additional results for ablation study on the automatic model selection below at Section 5.3.2.

Finally, note that in Table 4 we have also included the tuning time comparison between the *HNMFk Classifier* and the baseline models. The 5.77 days of tuning time listed for *HNMFk Classifier* comes from our performance analysis on selecting the cluster uniformity threshold t and understanding the effects of different hyper-parameter values of *NMFk*. We selected $t = 1$ for higher performance based on what we learned from the results of our experiments discussed in Section 5.1.1, and showed that the hyper-parameters of *NMFk* has a minimal affect on the model's performance in Section 5.1.3. Note that the k selection procedure of *HNMFk Classifier*, which comes from the *NMFk* algorithm, is not a hyper-parameter adjustment, but a model selection, which is integrated in the algorithm [38]; therefore, it is not included in the tuning time. Instead, it is reported as the model training time. Our method takes about 8 days to complete running, which is significantly longer than our baseline models. In comparison to the traditional ML methods (in our case, the baseline models used in the experiments), our method is not a fast predictor. Instead, *HNMFk Classifier* is a bulk-classification method. The aforementioned 8 days computation time is the total inference time for the *HNMFk Classifier*. Therefore, our model is not suitable for real-time solutions, that is, for analysing of a single specimen at the time it comes in the system. Our method rather can be used for an accurate malware classification early in the labeling process. We have investigated the use of latent signatures for a real-time solution in our follow-up work [23].

5.3 Ablation Studies

In our ablation studies we investigate the benefit of performing bulk classification and carefully choosing the number of clusters. To this end, during the first study we change the bulk classifier structure of our approach to form a more classical model, which we call the *HNMFk Classical Classifier*. During the second study, we ablate the automatic model selection heuristic from our method. The small subset of the EMBER-2018 dataset, as described in Section 4.2, is also used in our ablation studies in this section. As mentioned above, we use the top 10 malware families in the dataset with 1,000 specimens, each randomly sampled. Each experiment is run 10 times using a different random subset each time.

5.3.1 Bulk Classification. To show that there is a benefit to doing bulk classification for our methodology, we compare the performance of the *HNMFk Classifier* to the *HNMFk Classical Classifier*, a model that does not perform bulk classification. This model also uses the known samples to form the hierarchical graph. We then predict the unknown samples separately over the hierarchical graph by following the edges, and computing similarity scores at nodes. For each of the n unknown malware samples, we obtain the cluster assignment by comparing the features $X_{i:}$ (ith sample) to the rows of the latent factor $\mathbf{H}$ using *cosine-similarity* score:

$$\text{cluster}(i) = \underset{0 \le j \le k^{opt}}{\arg\max}\,(1 - \text{cosine-distance}(\mathbf{H}_{ji}, \mathbf{X}_{i:})) \tag{3}$$

We follow each sub-clusters, comparing the features vector for the ith sample to $\mathbf{H}$ at each step, until we reach a leaf where we predict the label of the specimen i in a semi-supervised fashion. In Figure 14 we compare the F1 scores obtained from our ablation studies to *HNMFk Classifier* as the fraction of unknown samples change. From the figure, it can be seen that performing classification with *HNMFk Classical Classifier* yields unstable results, and our method *HNMFk Classifier* outperforms this model. This shows that bulk classification is beneficial in obtaining stable and accurate inference results.

5.3.2 Determination of the Number of Clusters. The *HNMFk Classifier* utilizes the estimated number of components predicted by the *NMFk* algorithm to achieve good separability of malware families. For the next ablation study, we look at the benefit of estimating k, or the number of clusters. During this study, we form another classifier named the *HNMF2 Classifier*, based on the previous work of Gillis et al. [29], which chooses $k = 2$ at each node, i.e. separate the data into two clusters at each step, until each known sample falls in separate leaf nodes.

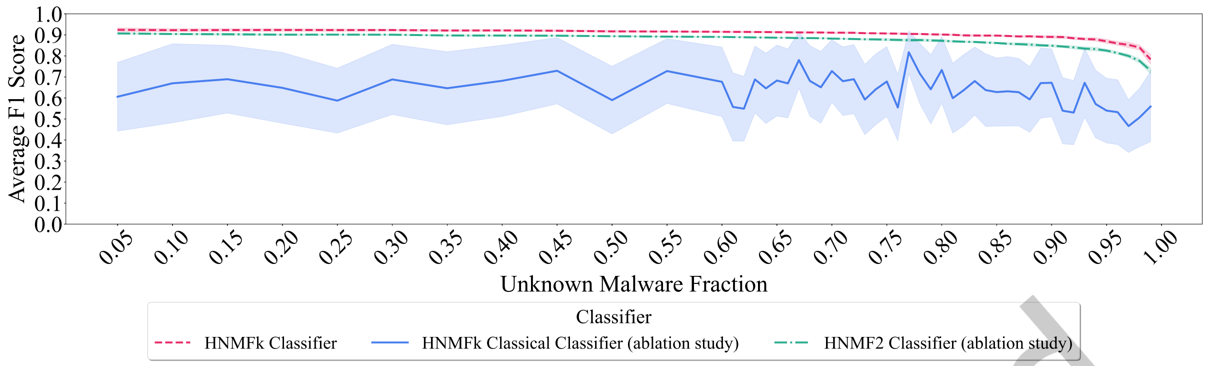

Fig. 14. The performance of the HNMFk Classifier is compared to the other variants of our method from the ablations studies, as the fraction of the unknown malware is changed.

In Figure 14, we also provide the results for the *HNMF2 Classifier*. Choosing $k = 2$ at each step performs almost as well as our approach. As also argued in [29], this result points out the benefit of hierarchical setting. Even if we make a bad separation of the samples due to rank-two factorization, the hierarchical approach will fix the separations in the proceeding splits. However, although slightly, our model outperforms the *HNMF2 Classifier*, which shows that choosing the number of components carefully using a heuristic is beneficial.

6 FUTURE WORK

The *HNMFk Classifier* has a significantly longer process time than the other ML methods which we used for comparison. The main cause for the increased computation time is the search of the number of clusters. *NMFk* performs this search in a sequential manner, where each value of k is tried, one after another. However, each rank k factorization is independent from one another. Therefore, future work can consider parallelization of this task, or a distributed version of this task utilizing High-Performance Computing (HPC) environments [12, 13, 16, 17].

Another future work includes the manual analysis of the specimens that fall in each cluster in the graph. It would be interesting to see which malware families are clustered together as we look at different nodes in the graph. This can help us understand if malware is clustered by type at first (such as botnet, backdoor, etc.), and then begin to separate into the families as we continue deeper in the graph. Future work can also include benign-ware as a class similar to [47, 50, 71].

We can also try to accelerate the computation time for clustering techniques via similarity-based approaches such as LZJD [57, 60] or BWMD [59] by using the *HNMFk Classifier* as a pre-processing step to obtain a hierarchical graph, where we then apply similarity comparisons only in the sub-trees instead of the entire dataset.

7 CONCLUSION

In this paper, we introduced a novel semi-supervised classifier named the *HNMFk Classifer*, that is capable of performing accurate bulk classification of thousands of malware families under extreme class imbalance conditions using the latent features extracted via *NMFk*, which is used to perform automatic model selection, i.e. to estimate the number of clusters. Our method's ability to perform abstaining predictions allows it to maintain its accuracy when using a small amount of labeled data and when performing inference over novel malware families. In our experiments, we classified Windows malware using static malware analysis based features. *HNMFk Classifer* is compared against the state-of-the-art baseline supervised and semi-supervised solutions, on which the prior work reported benchmarks, and surpassed their performance under the realistic experiment setting.

Our new solution can be used to assist reverse engineers and malware analysts in the labeling process of malware families, outside the real-time environments.

ACKNOWLEDGMENTS

This manuscript has been approved for unlimited release and has been assigned LA-UR-23-30350. We thank Nick Solovyev and Drew Barlow for helpful suggestions and edits. This research was partially funded by the Los Alamos National Laboratory (LANL) Laboratory Directed Research and Development (LDRD) grant 20190020DR and LANL Institutional Computing Program, supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001.

REFERENCES

[1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*. 183–194.

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

[3] BS. Alexandrov, LB. Alexandrov, and VG. Stanev et al. 2020. Source identification by non-negative matrix factorization combined with semi-supervised clustering. *US Patent S10,776,718* (2020).

[4] Boian Alexandrov, Velimir Vesselinov, and Kim Orskov Rasmussen. 2021. *SmartTensors Unsupervised AI Platform for Big-Data Analytics*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States). https://www.lanl.gov/collaboration/smart-tensors/LA-UR-21-25064.

[5] Ludmil B Alexandrov, Jaegil Kim, Nicholas J Haradhvala, Mi Ni Huang, Alvin Wei Tian Ng, Yang Wu, Arnoud Boot, Kyle R Covington, Dmitry A Gordenin, Erik N Bergstrom, et al. 2020. The repertoire of mutational signatures in human cancer. *Nature* 578, 7793 (2020), 94–101.

[6] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Samuel AJR Aparicio, Sam Behjati, Andrew V Biankin, Graham R Bignell, Niccolo Bolli, Ake Borg, Anne-Lise Børresen-Dale, et al. 2013. Signatures of mutational processes in human cancer. *Nature* 500, 7463 (2013), 415–421.

[7] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. 2013. Deciphering signatures of mutational processes operative in human cancer. *Cell reports* 3, 1 (2013), 246–259.

[8] H. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv* abs/1804.04637 (2018).

[9] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). arXiv:1804.04637 [cs.CR]

[10] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[11] Márton Bak, Dorottya Papp, Csongor Tamás, and Levente Buttyán. 2020. Clustering IoT Malware based on Binary Similarity. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–6.

[12] Manish Bhattarai, Ismael Boureima, Erik Skau, Benjamin Nebgen, Hristo Djidjev, Sanjay Rajopadhye, James P Smith, Boian Alexandrov, et al. 2023. Distributed non-negative rescal with automatic model selection for exascale data. *J. Parallel and Distrib. Comput.* 179 (2023), 104709.

[13] Manish Bhattarai, Ben Nebgen, Erik Skau, Maksim Eren, Gopinath Chennupati, Raviteja Vangara, Hristo Djidjev, John Patchett, Jim Ahrens, and Boian ALexandrov. 2021. pyDNMFk: Python Distributed Non Negative Matrix Factorization. https://github.com/lanl/pyDNMFk. https://doi.org/10.5281/zenodo.4722448

[14] Christopher M Bishop. 1999. Bayesian pca. *Advances in neural information processing systems* (1999), 382–388.

[15] K. Bissell and L. Ponemon. 2019. *The Cost of Cybercrime*. Technical Report. Accenture, Ponemon Institute. https://www.accenture.com/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf

[16] Ismael Boureima, Manish Bhattarai, Maksim Ekin Eren, Erik West Skau, Philip Romero, Stephan Johannes Eidenbenz, and Boian S. Alexandrov. 2022. Distributed out-of-memory NMF on CPU/GPU architectures. *The Journal of Supercomputing* (2022). https://api.semanticscholar.org/CorpusID:247011761

[17] Ismael Boureima, Manish Bhattarai, Maksim E Eren, Nick Solovyev, Hristo Djidjev, and Boian S Alexandrov. 2022. Distributed out-of-memory svd on cpu/gpu architectures. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–8.

[18] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. 2004. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences* 101, 12 (2004), 4164–4169.

[19] John Canny. 2004. GaP: a factor model for discrete data. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 122–129.

[20] Léna Carel and Pierre Alquier. 2021. Simultaneous dimension reduction and clustering via the NMF-EM algorithm. *Advances in Data Analysis and Classification* 15 (2021), 231–260.

[21] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[22] Thiago de Paulo Faleiros and Alneu de Andrade Lopes. 2016. On the equivalence between algorithms for Non-negative Matrix Factorization and Latent Dirichlet Allocation.. In *ESANN*.

[23] Maksim E Eren, Manish Bhattarai, Kim Rasmussen, Boian S Alexandrov, and Charles Nicholas. 2023. MalwareDNA: Simultaneous Classification of Malware, Malware Families, and Novel Malware. *arXiv preprint arXiv:2309.01350* (2023).

[24] Maksim E. Eren, Manish Bhattarai, Nicholas Solovyev, Luke E. Richards, Roberto Yus, Charles Nicholas, and Boian S. Alexandrov. 2022. One-Shot Federated Group Collaborative Filtering. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. 647–652. https://doi.org/10.1109/ICMLA55696.2022.00107

[25] Maksim E. Eren, Nick Solovyev, Manish Bhattarai, Kim Ø. Rasmussen, Charles Nicholas, and Boian S. Alexandrov. 2022. SeNMFk-SPLIT: Large Corpora Topic Modeling by Semantic Non-Negative Matrix Factorization with Automatic Model Selection. In *Proceedings of the 22nd ACM Symposium on Document Engineering* (San Jose, California) *(DocEng '22)*. Association for Computing Machinery, New York, NY, USA, Article 10, 4 pages. https://doi.org/10.1145/3558100.3563844

[26] External Data Source. 2018. VirusShare Dataset. https://doi.org/10.23721/100/1504313

[27] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. 2018. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. *IEEE Transactions on Information Forensics and Security* 13, 8 (2018), 1890–1905. https://doi.org/10.1109/TIFS.2018.2806891

[28] Cédric Févotte and A Taylan Cemgil. 2009. Nonnegative matrix factorizations as probabilistic inference in composite models. In *2009 17th European Signal Processing Conference*. IEEE, 1913–1917.

[29] Nicolas Gillis, Da Kuang, and Haesun Park. 2014. Hierarchical clustering of hyperspectral images using rank-two nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing* 53, 4 (2014), 2066–2078.

[30] Derek Greene, Derek O'Callaghan, and Pádraig Cunningham. 2014. How many topics? stability analysis for topic models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 498–513.

[31] Rachel Grotheer, Yihuan Huang, Pengyu Li, Elizaveta Rebrova, Deanna Needell, Longxiu Huang, Alona Kryshchenko, Xia Li, Kyung Ha, and Oleksandr Kryshchenko. 2020. COVID-19 Literature Topic-Based Search via Hierarchical NMF. *arXiv preprint arXiv:2009.09074* (2020).

[32] Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. 2016. An approach for detection and family classification of malware based on behavioral analysis. In *2016 International Conference on Computing, Networking and Communications (ICNC)*. 1–5. https://doi.org/10.1109/ICCNC.2016.7440587

[33] Simon Haykin. 1994. *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[34] Winston Haynes. 2013. *Wilcoxon Rank Sum Test*. Springer New York, New York, NY, 2354–2355. https://doi.org/10.1007/978-1-4419-9863-7_1185

[35] Wenyi Huang and Jay Stokes. 2016. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In *Proceedings of 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016)* (proceedings of 13th international conference on detection of intrusions and malware, and vulnerability assessment (dimva 2016) ed.). Springer, 399–418. https://www.microsoft.com/en-us/research/publication/mtnet-multi-task-neural-network-dynamic-malware-classification/

[36] IBM. 2021. *Cost of a Data Breach Report*. Technical Report. IBM. https://www.ibm.com/security/data-breach

[37] SM Ashiqul Islam, Marcos Díaz-Gay, Yang Wu, Mark Barnes, Raviteja Vangara, Erik N Bergstrom, Yudou He, Mike Vella, Jingwei Wang, Jon W Teague, et al. 2022. Uncovering novel mutational signatures by de novo extraction with SigProfilerExtractor. *Cell Genomics* (2022), 100179.

[38] SM Ashiqul Islam, Yang Wu, Marcos Díaz-Gay, Erik N Bergstrom, Yudou He, Mark Barnes, Mike Vella, Jingwei Wang, Jon W Teague, Peter Clapham, et al. 2021. Uncovering novel mutational signatures by de novo extraction with SigProfilerExtractor. *BioRxiv* (2021), 2020–12.

[39] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (oct 2002), 422–446. https://doi.org/10.1145/582415.582418

[40] Jianguo Jiang, Song Li, Min Yu, Gang Li, Chao Liu, Kai Chen, Hui Liu, and Weiqing Huang. 2019. Android malware family classification based on sensitive opcode sequence. In *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–7.

[41] Kaspersky. 2020. *Machine Learning Methods for Malware Detection*. Technical Report.

[42] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[43] Da Kuang and Haesun Park. 2013. Fast rank-2 nonnegative matrix factorization for hierarchical document clustering. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 739–747.

[44] Daniel D Lee and H Sebastian Seung. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401, 6755 (1999), 788–791.

[45] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52. http://jmlr.org/papers/v18/16-558.html

[46] Yeong Tyng Ling, Nor Fazlida Mohd Sani, Mohd Taufik Abdullah, and Nor Asilah Wati Abdul Hamid. 2019. Nonnegative matrix factorization and metamorphic malware detection. *Journal of Computer Virology and Hacking Techniques* 15, 3 (2019), 195–208.

[47] Nicola Loi, Claudio Borile, and Daniele Ucci. 2021. Towards an Automated Pipeline for Detecting and Classifying Malware through Machine Learning. *arXiv preprint arXiv:2106.05625* (2021).

[48] David JC MacKay et al. 1994. Bayesian nonlinear modeling for the prediction competition. *ASHRAE transactions* 100, 2 (1994), 1053–1062.

[49] Microsoft 365 Defender Threat Intelligence Team. 2020. *Microsoft researchers work with Intel Labs to explore new deep learning approaches for malware classification*. https://www.microsoft.com/security/blog.

[50] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. 2015. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security* 52 (2015), 251–266. https://doi.org/10.1016/j.cose.2015.04.001

[51] Morten Mørup and Lars Kai Hansen. 2009. Tuning pruning in sparse non-negative matrix factorization. In *2009 17th European Signal Processing Conference*. IEEE, 1923–1927.

[52] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security* (Pittsburgh, Pennsylvania, USA) *(VizSec '11)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2016904.2016908

[53] Benjamin T Nebgen, Raviteja Vangara, Miguel A Hombrados-Herrera, Svetlana Kuksova, and Boian S Alexandrov. 2021. A neural network for determination of latent dimensionality in non-negative matrix factorization. *Machine Learning: Science and Technology* 2, 2 (2021), 025012.

[54] Andre T Nguyen, Edward Raff, Charles Nicholas, and James Holt. 2021. Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints. *arXiv preprint arXiv:2108.04081* (2021).

[55] Bernardo Quintero. 2019. *VirusTotal += Bitdefender Theta*. https://blog.virustotal.com/2019/10/virustotal-bitdefender-theta.html

[56] Bernardo Quintero. 2019. *VirusTotal += Sangfor Engine Zero*. https://blog.virustotal.com/2019/11/virustotal-sangfor-engine-zero.html

[57] Edward Raff and Charles Nicholas. 2017. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. ACM, New York, NY, USA, 1007–1015. https://doi.org/10.1145/3097983.3098111

[58] Edward Raff and C. Nicholas. 2020. A Survey of Machine Learning Methods and Challenges for Windows Malware Classification. *ArXiv* abs/2006.09271 (2020).

[59] Edward Raff, Charles Nicholas, and Mark McLean. 2020. A New Burrows Wheeler Transform Markov Distance. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*. http://arxiv.org/abs/1912.13046

[60] Edward Raff and Charles K. Nicholas. 2018. Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash. *Digital Investigation* (feb 2018). https://doi.org/10.1016/j.diin.2017.12.004

[61] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[62] Tian Shi, Kyeongpil Kang, Jaegul Choo, and Chandan K Reddy. 2018. Short-text topic modeling via non-negative matrix factorization enriched with local word-context correlations. In *Proceedings of the 2018 World Wide Web Conference*. 1105–1114.

[63] Bowen Sun, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. 2017. Malware family classification method based on static feature extraction. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. 507–513. https://doi.org/10.1109/CompComm.2017.8322598

[64] Vincent YF Tan and Cédric Févotte. 2012. Automatic relevance determination in nonnegative matrix factorization with the/spl beta/-divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 7 (2012), 1592–1605.

[65] The Independent IT Security Institute. 2021. Malware Statistics & Trends Report: AV-TEST. https://www.av-test.org/en/statistics/malware/

[66] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. 2014. A deep semi-nmf model for learning hidden representations. In *International Conference on Machine Learning*. PMLR, 1692–1700.

[67] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html

[68] Raviteja Vangara, Manish Bhattarai, Erik Skau, Gopinath Chennupati, Hristo Djidjev, Thomas Tierney, James P Smith, Valentin G Stanev, and Boian S Alexandrov. 2021. Finding the Number of Latent Topics with Semantic Non-negative Matrix Factorization. *IEEE Access* (2021).

[69] Raviteja Vangara, Erik Skau, Gopinath Chennupati, Hristo Djidjev, Thomas Tierney, James P Smith, Manish Bhattarai, Valentin G Stanev, and Boian S Alexandrov. 2020. Semantic nonnegative matrix factorization with automatic model determination for topic modeling. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 328–335.

[70] Raviteja Vangara, Erik Skau, Gopinath Chennupati, Hristo Djidjev, Thomas Tierney, James P. Smith, Manish Bhattarai, Valentin G. Stanev, and Boian S. Alexandrov. 2020. Semantic Nonnegative Matrix Factorization with Automatic Model Determination for Topic Modeling. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 328–335. https://doi.org/10.1109/ICMLA51294.2020.00060

[71] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. 2019. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access* 7 (2019), 46717–46738. https://doi.org/10.1109/ACCESS.2019.2906934

[72] Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. 267–273.

[73] David Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics* (Cambridge, Massachusetts) *(ACL '95)*. Association for Computational Linguistics, USA, 189–196. https://doi.org/10.3115/981658.981684

[74] Lijun Zhang, Zhengguang Chen, Miao Zheng, and Xiaofei He. 2011. Robust non-negative matrix factorization. *Frontiers of Electrical and Electronic Engineering in China* 6, 2 (2011), 192–200.

[75] Shao-Huai Zhang, Cheng-Chung Kuo, and Chu-Sing Yang. 2019. Static PE Malware Type Classification Using Machine Learning Techniques. In *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)*. 81–86. https://doi.org/10.1109/ICEA.2019.8858297

[76] Yanxin Zhang, Yulei Sui, Shirui Pan, Zheng Zheng, Baodi Ning, Ivor Tsang, and Wanlei Zhou. 2020. Familial Clustering for Weakly-Labeled Android Malware Using Hybrid Representation Learning. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3401–3414. https://doi.org/10.1109/TIFS.2019.2947861