DDoS attack detection via Boolean Matrix Factorization

Veronika Pavlikova*, Martin Trnecka

Department of Computer Science, Faculty of Science, Palacký University Olomouc, Czech Republic

Abstract

Distributed Denial-of-Service (DDoS) attacks remain among the most damaging threats to modern network infrastructures, and their detection increasingly relies on machine learning. Many high-performing detectors, however, behave as black boxes and provide little insight that security analysts can turn into actionable detection rules. In this paper, we investigate Boolean Matrix Factorization (BMF) as an interpretable alternative for DDoS attack detection. We propose a factor-based classification method that factorizes the data separately for each class, yielding compact, class-specific factors that translate directly into human-readable rules. The approach is evaluated on the CIC-DDoS2019 dataset and compared with decision trees, the Naive Bayes classifier, and support vector machines. The results show that factor-based classification attains accuracy comparable to these conventional methods while producing substantially fewer and more transparent rules, which makes it well suited for practical deployment in detection systems.

Keywords

DDoS, Boolean matrix factorization, classification, interpretability, network intrusion detection

1. Introduction

Distributed Denial-of-Service (DDoS) attacks represent one of the most persistent and damaging threats in modern computer networks. By overwhelming network resources with large volumes of malicious traffic, DDoS attacks can render critical services unavailable, resulting in substantial financial losses, reputational damage, and service disruption. The increasing dependence of society on online services, cloud infrastructures, Internet-of-Things (IoT) ecosystems, and software-defined networks (SDN) has further amplified the importance of effective DDoS detection and mitigation mechanisms. Recent studies indicate that DDoS attacks continue to grow both in frequency and sophistication, making their timely detection a crucial cybersecurity challenge.

Consequently, a considerable amount of research effort has been devoted to the development of DDoS detection methods. Traditional approaches based on statistical analysis, signatures, or manually designed rules have gradually been complemented and, in many cases, replaced by machine learning techniques. Recent surveys show that neural networks, decision trees or support vector machines provide very good performance [1, 2].

Despite this, many state-of-the-art approaches suffer from a significant limitation: lack of interpretability. Deep learning models often operate as black boxes, providing little insight into the patterns that distinguish malicious traffic from legitimate network communication. While high predictive accuracy is important, practical deployment in cybersecurity environments frequently requires additional transparency. Security analysts and network administrators must understand why a particular flow or connection has been classified as malicious and should be able to derive actionable knowledge that can be incorporated into detection rules, firewall policies, or intrusion detection systems. The need for explainable and trustworthy cybersecurity solutions has been repeatedly highlighted (see, e.g., [3, 2]).

Most of the recent progress in this area has been driven by deep neural networks. Although such models frequently achieve excellent detection accuracy, their opacity limits their usefulness in operational settings, where analysts need to understand, audit, and act upon detection decisions. For this reason, we deliberately do not focus on deep learning and instead concentrate on methods that are both

ITAT 2026: Information Technologies – Applications and Theory, 2026

*Corresponding author.

veronika.pavlikova@upol.cz (V. Pavlikova); martin.trnecka@gmail.com (M. Trnecka)

0009-0001-5686-6044 (V. Pavlikova); 0000-0001-7770-2033 (M. Trnecka)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-09

CEUR-WS.org/Vol-4323/paper35.pdf

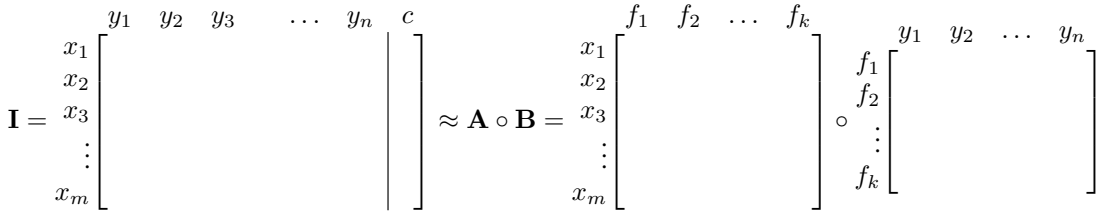


Figure 1: Basic Boolean matrix factorization model considered in this paper. The operation ∘ denotes Boolean matrix multiplication. The symbol ≈ indicates that the factorization need not be exact, i.e., $\mathbf{I}$ and $\mathbf{A} \circ \mathbf{B}$ may differ in some entries.

accurate and practically usable, i.e., methods whose output can be directly inspected by a human, e.g., decision trees, and turned into explicit detection rules.

In this paper, we investigate the use of Boolean Matrix Factorization (BMF) for DDoS attack detection. Unlike many existing machine learning approaches, BMF decomposes data into a set of interpretable binary factors that describe recurring patterns within the analyzed traffic. These factors can be directly inspected and interpreted by domain experts, providing valuable insight into the structure of both benign and malicious traffic. Furthermore, the discovered factors can be naturally transformed into human-readable rules, facilitating their integration into practical detection and mitigation systems.

The rest of the paper is organized as follows. Section 2 introduces the necessary background on Boolean Matrix Factorization. Section 3 presents the proposed factor-based detection method. Section 4 reports the experimental evaluation on the CIC-DDoS2019 dataset and discusses the construction of detection rules. Finally, Section 5 concludes the paper and outlines directions for future work.

2. Preliminaries

Boolean Matrix Factorization (BMF), also known as Boolean matrix decomposition, is a machine learning technique commonly used for the analysis of binary object–attribute data. It reveals latent patterns, referred to as factors, that capture meaningful structures within the data and provide an interpretable description of the underlying relationships between objects and attributes.

The principle behind BMF is straightforward. Given an input binary matrix $\mathbf{I}$ of size $m \times n$, representing object–attribute data, BMF aims to decompose it into two binary matrices: $\mathbf{A}$, an $m \times k$ object–factor matrix, and $\mathbf{B}$, a $k \times n$ factor–attribute matrix. Each factor can be interpreted as a latent pattern describing a subset of objects sharing a subset of attributes. The BMF model is captured in Figure 1. The Boolean product $\circ$ of $\mathbf{A}$ and $\mathbf{B}$, which is simply defined as

$$(\mathbf{A} \circ \mathbf{B})_{ij} \approx \bigvee_{l=1}^{k} (\mathbf{A}_{il} \wedge \mathbf{B}_{lj}),$$

reconstructs or approximates the original matrix $\mathbf{I}$.

In practice, exact reconstruction is often undesirable due to the presence of noise and inconsistencies in real-world data [4]. Therefore, the objective is typically to find matrices $\mathbf{A}$ and $\mathbf{B}$ such that the Boolean product $\mathbf{A} \circ \mathbf{B}$ approximates $\mathbf{I}$ while using as few factors as possible. In such case the resulting factors provide a compact and interpretable representation of the original data.

Geometrically, each factor corresponds to a rectangle in the data matrix—a submatrix consisting primarily of ones (and, in some cases, a small number of zeros)—that becomes clearly visible after a suitable permutation of the rows and columns [5].

Over the years, numerous algorithms for Boolean Matrix Factorization have been developed [6]. Two of the most prominent representatives are GreConD [7], which is based on Formal Concept Analysis, and Asso [8], which relies on association-based pattern discovery.

Besides the matrix view, we also use the set-theoretic notation common in Formal Concept Analysis (FCA) [9], on which one of the factorization algorithms mentioned above is based. The input matrix $\mathbf{I}$ is regarded as a binary relation between a set of objects $X = \{x_1, \ldots, x_m\}$ and a set of attributes $Y = \{y_1, \ldots, y_n\}$, where $\mathbf{I}_{ij} = 1$ means that object x_i has attribute y_j. We identify an object x with the set of attributes it possesses, i.e., the corresponding row of $\mathbf{I}$, and we write $|x|$ for the number of attributes of x. Accordingly, each factor f_l is represented as a pair $\langle a, b\rangle$, where $a \subseteq X$ is its *extent*—the set of objects in which the factor occurs, i.e., the l-th column of $\mathbf{A}$—and $b \subseteq Y$ is its *intent*—the set of attributes the factor comprises, i.e., the l-th row of $\mathbf{B}$. The factor thus covers exactly the submatrix $a \times b$ of $\mathbf{I}$, and we write $|a|$ and $|b|$ for the number of objects in the extent and attributes in the intent, respectively. In FCA-based factorization the factors correspond to formal concepts [7], i.e., maximal rectangles full of 1s in $\mathbf{I}$. Set operations apply directly between an intent and an object: for an intent b and an object x, $b \cap x$ denotes the attributes shared by b and x, and $b \subseteq x$ means that x has all attributes of b.

In the classification setting considered in this paper, every object is additionally labeled by a *class attribute* $c \in \{1, 2, \ldots, l\}$, shown as the separate column c in Figure 1. This attribute records the class membership of each object and is not part of the relation being factorized. We denote by c_i the i-th class and by $|c_i|$ the number of objects belonging to it.

3. DDoS attack detection method

Since factors provide a compact yet expressive explanation of the data, they can be naturally employed for classification tasks. This idea was originally introduced in [7], where a method for transforming the original attribute space into a factor space, and vice versa, was proposed. Later, this approach was utilized in [10] and [11], where it was demonstrated that such a transformation can improve classification performance.

One limitation of this approach is that factors may simultaneously apply to objects belonging to different classes. Although this issue can be partially addressed by assigning the factor to the most frequently represented class, such factors become less useful in scenarios involving a larger number of classes. Typically, these correspond to overly general factors capturing many shared characteristics across classes. As a result, they are often not informative for the classification task itself and may even become misleading.

To overcome this limitation, we introduced a novel factor-based classification approach. Instead of computing all factors from the complete dataset $\mathbf{I}$, we partition the data according to the class attribute $c \in \{1, 2, 3, \ldots, l\}$ into several smaller datasets $\mathbf{I_1}, \mathbf{I_2}, \ldots, \mathbf{I_l}$. Factors are then computed independently for each dataset.

Several important observations should be emphasized here. The factors obtained in this way differ from those computed from the complete dataset. In particular, each factor is associated exclusively with objects belonging to a single class, while factors spanning multiple classes disappear entirely. This property makes the resulting factors more class-specific and therefore more suitable for classification tasks.

Another advantage of the proposed approach is computational efficiency. Since factorization is performed on smaller datasets, the computation is significantly faster compared to factorization of the original complete dataset.

The extracted factors can be used for classification as follows. For a classified object x, we measure how well the factors obtained from each class dataset $\mathbf{I}_i$ apply to it, and assign x to the class whose factors match it best. We make this notion precise in the rest of this section.

The degree of applicability of a factor $\langle a, b\rangle$ to an object x can be computed as $\frac{|b \cap x|}{|b|}$, which naturally captures how large a part of the factor (its intent) is reflected in the given object. The notion of applicability is, however, considerably more involved and depends on a number of aspects, such as the importance of an attribute or its distinctiveness—that is, how characteristic the attribute is of a given class. Such information is usually difficult to obtain, as it requires additional knowledge about the data.

Nevertheless, from the data alone we can estimate the probability of an attribute's occurrence within a given class, which captures this aspect at least to a limited extent.

To capture this, we attach to each factor a weight expressing how strongly it is associated with its class. The weight is derived from Bayesian statistics as

$$P(c_i \mid x) = \frac{P(x \mid c_i) \cdot P(c_i)}{P(x)}.$$

We thus examine the probability that an object x belongs to a given class c_i. The term $P(c_i)$ is computed simply as the number of occurrences of class c_i divided by the total number of objects in the data (i.e., the number of rows). We interpret $P(x \mid c_i)$ as the fraction of objects of class c_i that are covered by the factor $\langle a, b\rangle$ taken from the class dataset $\mathbf{I}_i$; it is evaluated as the size of the extent $|a|$ divided by the size of class c_i—that is, how strongly the factor is associated with the given class. Finally, $P(x)$ denotes the probability of occurrence of the factor's intent in the data, i.e., the number of rows with $b \subseteq x$ divided by the total number of rows.

In principle, the applicability degree introduced above could be used directly, with each factor contributing in proportion to how strongly it applies to x. Since this work is an initial study, we deliberately adopt a simpler variant: a factor $\langle a, b\rangle$ is considered applicable to x only when it applies fully, i.e., when $b \subseteq x$ (equivalently, when its applicability degree $|b \cap x|/|b|$ equals 1). The graded treatment, together with the attribute-importance refinement mentioned above, is left for a follow-up of this work.

Each factor that applies to x contributes its weight $P(c_i \mid x)$, and the score of class i is obtained as the sum of these contributions over the factors of the class. The object x is then assigned to the class with the highest score. This aggregation does not need to involve all factors of a class; it is sufficient to consider a small fixed number of the most prominent factors per class. Besides keeping the resulting model compact and easy to interpret, this further speeds up classification, since only a few factors per class are evaluated for each object. If no factor applies to x, the object is left unclassified; we return to this case in Section 4.3.

Example. To illustrate the method, consider a small object–attribute data set with five attributes $y_1, \ldots, y_5$ and six objects belonging to two classes ($c = 1$ and $c = 2$):

	y_1	y_2	y_3	y_4	y_5	c
x_1	1	1	1			1
x_2	1	1	1			1
x_3			1	1		1
x_4			1		1	2
x_5			1		1	2
x_6			1		1	2

Following the proposed approach, we factorize each class separately. For class 1 (objects x_1, x_2, x_3) we obtain two factors, $f_1 = \langle\{x_1, x_2\}, \{y_1, y_2, y_3\}\rangle$ and $f_2 = \langle\{x_3\}, \{y_3, y_4\}\rangle$, while class 2 (objects x_4, x_5, x_6) is described by a single factor $g_1 = \langle\{x_4, x_5, x_6\}, \{y_3, y_5\}\rangle$. We stress that the method does not depend on the particular factorization algorithm; any Boolean matrix factorization method, such as GreConD or Asso, can be used to obtain the factors.

Notice that attribute y_3 occurs in every object of both classes. On the complete data set it would therefore yield a large but non-discriminative factor spanning both classes. Under class-wise factorization, however, y_3 never appears on its own; it is absorbed into the class-specific factors, where it is combined with the attributes characteristic of the respective class.

For clarity, we first consider only the single most prominent factor of each class, namely f_1 for class 1 and g_1 for class 2, and classify a new object $x = \{y_1, y_2, y_3\}$. The intent of f_1 is fully contained in x, since $\{y_1, y_2, y_3\} \subseteq x$, so f_1 applies and contributes to class 1. The intent of g_1 is not contained in x

(the attribute y_5 is missing, and $|\{y_3, y_5\} \cap x|/|\{y_3, y_5\}| = 1/2 < 1$), so g_1 does not apply. Only class 1 receives a non-zero score, and x is assigned to class 1, as expected.

The method is, however, not restricted to a single factor per class, and considering more factors typically improves *coverage*, i.e., the proportion of the 1s in the data matrix that the factors account for; as a consequence, more objects match at least one factor and can be assigned a class. Consider the object $z = \{y_3, y_4\}$. Using only the most prominent factor of each class, neither f_1 (which requires y_1 and y_2) nor g_1 (which requires y_5) applies, so z would be left unclassified. Once the second factor of class 1 is taken into account, its intent $\{y_3, y_4\} \subseteq z$ is satisfied, f_2 applies, and z is correctly assigned to class 1. This illustrates how using a small number of factors per class, rather than a single one, increases the number of objects the classifier can handle.

4. Experimental evaluation

In this section, we evaluate the proposed factor-based classification method on real-world DDoS traffic. We first describe the CIC-DDoS2019 dataset and the preprocessing required to turn it into binary object–attribute data. We then compare the method against three conventional classifiers, reporting accuracy, precision, and recall across all traffic classes for both factorization algorithms considered. Finally, we discuss the practical benefit that motivates our approach, namely the construction of compact and interpretable detection rules.

4.1. Data

To demonstrate our proposed method, we selected the CIC-DDoS2019 dataset [12]. CIC-DDoS2019 is a publicly available dataset created by the Canadian Institute for Cybersecurity, which is widely used for training in DDoS attack detection and mitigation. It contains generated realistic network traffic with several types of DDoS attacks.

The dataset is divided into two main folders: one containing all traffic captured during the training phase and the other containing traffic captured during the testing phase. Within each folder, the data is further separated into multiple CSV files according to the type of DDoS attack represented. Each row in these files corresponds to a single network flow. The CSV files were generated using the *CICFlowMeter-V3* tool [13] and are suitable for direct use in machine learning tasks. However, before training the models, the data had to be merged and preprocessed. After merging all CSV files from the training and testing subsets, two datasets containing several million records were obtained. Due to the large size of the datasets and the significant class imbalance between DDoS attacks and benign traffic, both datasets had to be reduced.

First, the number of network flows in each CSV file was analyzed in order to balance the representation of individual DDoS attack types. Most files contained a similar number of network flows; however, the file containing the TFTP attack represented nearly half of the entire training dataset. Therefore, this file was reduced by 50%. Next, the ratio between benign and DDoS traffic was examined. The dataset was highly imbalanced, with benign traffic accounting for only approximately 0.11% of all records. To address this issue, the class distribution was adjusted to a 1:3 ratio between benign and DDoS traffic, with a maximum of 5,000 benign samples taken from each file. This ensured that the models were trained on a sufficient number of samples from both classes. The final number of samples from individual DDoS attacks can be seen in Table 1.

Table 1
Number of samples of individual types of DDoS attacks in training dataset.

BENIGN	DNS	NTP	SNMP	SSDP	LDAP	MYSSQL	NETBIOS	SYN	TFTP	UDP	UDPLAG
26004	9879	14909	4513	2251	4762	5853	4947	1017	14576	6391	9984

After preprocessing and removing all records containing missing values from both datasets, the resulting data consisted of more than 100,000 records with 88 attributes. To apply the proposed method, attributes containing predominantly missing or non-informative values were removed (e.g. *flow_id, source_ip*, 27 attributes in total), while the remaining attributes were converted into binary form. The conversion was performed differently depending on the attribute type and the information carried by each column. Attribute values were divided either into two categories based on a predefined criterion (e.g., *source_port* and *destination_port*), into three categories according to the maximum value in the column (at 1/3, 2/3, and the maximum value), or into four categories according to quartiles. The selected method was carefully chosen with respect to the nature of the data in each column. After preprocessing and binarization, the dataset contained 205 attributes and was ready for direct use.

4.2. Experimental evaluation

All experiments were conducted in MATLAB R2025b. The performance of the proposed approach was evaluated against three conventional machine learning methods: decision trees, the Naive Bayes classifier, and support vector machines. The corresponding models were implemented using the `fitctree`, `fitcnb`, and `fitcecoc` functions available in the MATLAB library. For factor-based classification, two binary matrix decomposition methods, namely *GreConD* [7] and *Asso* [8], were employed. The evaluation was carried out using three performance metrics: accuracy, precision, and recall.

Two versions of the dataset were used in the evaluation. In the first configuration, the original training dataset was partitioned into training and testing subsets using a 9:1 ratio. In the second configuration, the complete training dataset was used for model training, while the separate test dataset was reserved exclusively for evaluation.

For factor-based classification, determining the number of factors per class k used for comparison with the test data was a critical step. The number of factors was intentionally kept as low as possible for practical considerations. During the selection, we also monitored how often the classifier was unable to assign a class label (see Table 3). We evaluated all values of k from 1 to 10. The best performance was obtained using 3 factors per class: fewer factors led to underfitting, whereas a larger number of factors resulted in overfitting.

The results for classification on the split training data are presented in Table 2. As shown, classification based on binary factors yields strong performance, regardless of whether the GreConD or Asso factorization method is used. When compared to classical classification approaches, the factorization-based methods achieve comparable results. The classifier using the GreConD algorithm performs poorly in detecting LDAP and SSDP attacks, which is also observed for the other classifiers. In contrast, a notable improvement over the Naive Bayes classifier can be observed, particularly in the detection of benign traffic and NTP, TFTP, and UDPLAG attacks. Very similar pattern is observed for the classifier based on the Asso algorithm.

4.3. Discussion

In the previous section, we demonstrated that factor-based detection achieves performance comparable to selected machine learning methods. In this section, we focus on its practical application in the construction of rules for real-world detection systems.

Obtaining such rules from machine learning models is not always straightforward. For models such as Naive Bayes classifiers or support vector machines, extracting rules suitable for deployment in detection systems may be either highly complex or practically infeasible. In contrast, rules can be derived relatively easily from decision trees through inspection of the tree structure, where each branch corresponds to a single rule. Despite this apparent simplicity, several practical challenges must be considered. In particular, the complexity of the decision tree, reflected in its depth and number of branches, plays an important role. A tree with a large number of branches requires substantial effort to analyze and convert into detection rules. For illustration, an unconstrained decision tree trained on the

Table 2
Classification results on split training data.

Class	Metric	Decision Trees	Naive Bayes	SVM	GreConD (3 factors)	Asso (3 factors)
BENIGN	Accuracy	1.00	0.94	1.00	0.99	0.94
	Precision	1.00	0.97	1.00	0.95	0.82
	Recall	1.00	0.80	1.00	1.00	1.00
DNS	Accuracy	0.94	0.93	0.94	0.94	0.93
	Precision	0.63	0.58	0.63	0.61	0.60
	Recall	0.81	0.69	0.81	0.83	0.78
NTP	Accuracy	1.00	0.99	1.00	1.00	0.98
	Precision	1.00	0.99	1.00	1.00	1.00
	Recall	0.99	0.95	0.99	0.98	0.88
SNMP	Accuracy	0.96	0.94	0.96	0.96	0.96
	Precision	0.52	0.41	0.52	0.51	0.52
	Recall	0.72	0.72	0.72	0.61	0.61
SSDP	Accuracy	0.98	0.97	0.98	0.98	0.98
	Precision	1.00	0.13	0.00	0.00	0.00
	Recall	0.00	0.07	0.00	0.00	0.00
LDAP	Accuracy	0.95	0.95	0.95	0.95	0.95
	Precision	0.27	0.00	0.27	0.00	0.21
	Recall	0.01	0.00	0.01	0.00	0.01
MSSQL	Accuracy	0.99	0.91	0.99	0.98	0.98
	Precision	0.97	0.35	0.97	0.94	0.74
	Recall	0.84	0.82	0.84	0.61	0.86
NETBIOS	Accuracy	0.99	0.98	0.99	0.99	0.99
	Precision	0.91	0.70	0.91	0.85	0.91
	Recall	0.98	0.98	0.98	0.99	0.96
SYN	Accuracy	0.99	0.93	0.99	0.99	0.98
	Precision	0.00	0.10	0.00	0.07	0.10
	Recall	0.00	0.78	0.00	0.04	0.18
TFTP	Accuracy	0.99	0.94	0.99	0.98	0.97
	Precision	0.96	0.97	0.96	0.89	0.98
	Recall	0.99	0.60	0.99	0.99	0.83
UDP	Accuracy	0.96	0.95	0.96	0.96	0.96
	Precision	0.62	0.64	0.62	0.61	0.62
	Recall	0.96	0.48	0.97	0.97	0.88
UDPLAG	Accuracy	0.98	0.90	0.98	0.96	0.94
	Precision	0.89	0.48	0.89	0.90	0.80
	Recall	0.87	0.22	0.87	0.72	0.53

split dataset produces 137 leaves, corresponding to 137 distinct rules. Similarly, the tree generated for the second dataset configuration contains 140 leaves. Constructing and maintaining such a large number of rules is not only time-consuming, but can also significantly reduce the clarity and manageability of the overall detection system. While the number of leaves in a decision tree can be limited, our experiments demonstrated that this reduction may negatively affect classification performance.

In contrast, detection rules can be derived directly from Boolean factors by examining their attributes. Since each factor is defined by its objects and attributes, the corresponding attributes can be directly inspected to construct a detection rule. To make this concrete, consider the factor $f_1 = \langle \{x_1, x_2\}, \{y_1, y_2, y_3\} \rangle$ of class 1 from the example in Section 3. Its intent translates directly into

the rule *if an object has attributes y_1, y_2, and y_3, then it belongs to class* 1. In general, each factor $\langle a, b\rangle$ of class i yields a rule whose antecedent is the conjunction of the attributes in its intent b and whose consequent is class i. Because the attributes are binarized network-flow features, such a rule corresponds to a human-readable condition on the flow (e.g., a combination of port, duration, or packet-rate ranges) that can be inserted directly into a detection system. Their number is also inherently limited from the outset. Furthermore, the proposed approach generates factors separately for each class and organizes them in a structured matrix representation. This significantly improves the readability and interpretability of the resulting rules, particularly when compared to decision trees.

A practical limitation of the proposed method should also be mentioned. Because a factor contributes to the classification only when its intent is fully contained in the object, some test objects may not match any factor and are therefore left unclassified. This effect is most pronounced when only a few factors per class are used, since a small set of factors covers fewer of the 1s in the data; as a consequence, more objects match no factor and are left unclassified. Increasing the number of factors per class reduces the number of unclassified objects, but at the cost of a larger and less compact rule set. Table 3 reports the number of unclassified test objects for different numbers of factors per class k, and illustrates this trade-off between coverage and model compactness.

Table 3
Number and percentage share of test objects left unclassified (i.e., no factor applies) for different numbers of factors per class k.

k	GreConD		Asso	
1	2647	25.2%	3900	37.1%
2	790	7.5%	2737	26.0%
3	178	1.7%	1588	15.1%
4	102	1.0%	1306	12.5%
5	92	0.9%	964	9.2%
6	52	0.5%	446	4.2%
7	45	0.4%	426	4.1%
8	44	0.4%	357	3.4%
9	43	0.4%	302	2.9%
10	36	0.3%	222	2.1%

The number of unclassified objects decreases monotonically with k for both algorithms, since adding further factors can only increase coverage. The decrease is steepest for the first few factors and then levels off: for GreConD it drops from 2647 at $k = 1$ to 178 at $k = 3$ and only marginally further afterwards, reaching 36 at $k = 10$. Asso leaves considerably more objects unclassified across the whole range (e.g., 1588 at $k = 3$ and still 222 at $k = 10$). This indicates that GreConD covers the data with far fewer factors than Asso, which is consistent with its stronger classification performance reported above. The difference can be attributed to the nature of the two factorizations. Unlike GreConD, whose factors are formal concepts covering only 1s of $\mathbf{I}$, Asso does not restrict the factors in this way and may also cover some 0s, i.e., it commits an overcovering error [5]. The resulting factors describe the data only approximately, so their intents are matched exactly by fewer objects; this increases the number of unclassified objects and, in turn, degrades the classification performance of the Asso-based classifier.

5. Conclusions and further research

In this paper, we investigated the use of Boolean Matrix Factorization for the detection of DDoS attacks. We proposed a factor-based classification method that computes factors separately for each class. This design makes the resulting factors class-specific and avoids overly general factors that span several classes, which improves both their interpretability and their usefulness for classification.

Experiments on the CIC-DDoS2019 dataset showed that the proposed approach achieves detection performance comparable to that of established classifiers such as decision trees, the Naive Bayes classifier,

and support vector machines. Its decisive practical advantage, however, lies in interpretability: the discovered factors can be transformed into a small and well-organized set of human-readable detection rules, in contrast to the large number of branches produced by an unconstrained decision tree. This makes the method particularly attractive for deployment in real-world detection systems, where rules must be inspected, maintained, and trusted by human analysts.

The evaluation also revealed limitations. Certain attack types, such as SSDP, LDAP, and SYN, remain difficult to detect, a behaviour shared by the conventional classifiers as well. In future work, we plan to refine the applicability measure, in particular the proposed Bayesian formulation, to better account for the importance and distinctiveness of individual attributes. We further intend to automate the binarization step, to evaluate the method on additional datasets, and to study the automatic generation of deployable detection rules directly from the extracted factors.

Acknowledgments

This work was supported by Grant No. IGA_PrF_2026_26 of the Internal Grant Agency of Palacký University Olomouc.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT-5 and Grammarly in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] D. Akgun, S. Hizal, U. Cavusoglu, A new DDoS attacks intrusion detection model based on deep learning for cybersecurity, Computers & Security 118 (2022) 102748. doi:10.1016/j.cose.2022.102748.

[2] V. Hnamte, A. A. Najar, H. Nhung-Nguyen, J. Hussain, M. N. Sugali, DDoS attack detection and mitigation using deep neural network in SDN environment, Computers & Security 138 (2024) 103661. doi:10.1016/j.cose.2023.103661.

[3] A. A. Wabi, I. Idris, O. M. Olaniyi, J. A. Ojeniyi, DDoS attack detection in SDN: Method of attacks, detection techniques, challenges and research gaps, Computers & Security 139 (2024) 103652. doi:10.1016/j.cose.2023.103652.

[4] R. Belohlavek, M. Trnecka, Handling noise in Boolean matrix factorization, International Journal of Approximate Reasoning 96 (2018) 78–94. doi:10.1016/j.ijar.2018.03.006.

[5] R. Belohlavek, M. Trnecka, From-below approximations in Boolean matrix factorization: Geometry and new algorithm, Journal of Computer and System Sciences 81 (2015) 1678–1697. doi:10.1016/j.jcss.2015.06.002.

[6] P. Miettinen, S. Neumann, Recent developments in Boolean matrix factorization, in: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI), 2020, pp. 4922–4928. doi:10.24963/ijcai.2020/685.

[7] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, Journal of Computer and System Sciences 76 (2010) 3–20. doi:10.1016/j.jcss.2009.05.002.

[8] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE Transactions on Knowledge and Data Engineering 20 (2008) 1348–1362. doi:10.1109/TKDE.2008.53.

[9] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, 2 ed., Springer, 2024. doi:10.1007/978-3-031-63422-2.

[10] R. Belohlavek, J. Outrata, M. Trnecka, Impact of Boolean factorization as preprocessing methods for classification of Boolean data, in: L. Szathmary, U. Priss (Eds.), Proceedings of the 9th International Conference on Concept Lattices and Their Applications (CLA 2012), Fuengirola (Málaga), Spain, 2012, pp. 305–316.

[11] R. Belohlavek, J. Outrata, M. Trnecka, Impact of Boolean factorization as preprocessing methods for classification of Boolean data, Annals of Mathematics and Artificial Intelligence 72 (2014) 3–22. doi:10.1007/s10472-014-9414-x.

[12] I. Sharafaldin, A. H. Lashkari, S. Hakak, A. A. Ghorbani, Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy, in: 2019 International Carnahan Conference on Security Technology (ICCST), 2019, pp. 1–8. doi:10.1109/CCST.2019.8888419.

[13] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, A. A. Ghorbani, Characterization of Tor traffic using time based features, in: Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP), SciTePress, 2017, pp. 253–262. doi:10.5220/0006105602530262.