Bayesian Networks for Interpretable Cyberattack Detection

Barnett Yang
Sandia National Laboratories
bzyang@sandia.gov

Matthew Hoffman
Sandia National Laboratories
mjhoffm@sandia.gov

Nathanael Brown
Sandia National Laboratories
njbrown@sandia.gov

Abstract

The challenge of cyberattack detection can be illustrated by the complexity of the MITRE ATT&CK^{TM} matrix, which catalogues >200 attack techniques (most with multiple sub-techniques). To reliably detect cyberattacks, we propose an evidence-based approach which fuses multiple cyber events over varying time periods to help differentiate normal from malicious behavior. We use Bayesian Networks (BNs) – probabilistic graphical models consisting of a set of variables and their conditional dependencies – for fusion/classification due to their interpretable nature, ability to tolerate sparse or imbalanced data, and resistance to overfitting. Our technique utilizes a small collection of expert-informed cyber intrusion indicators to create a hybrid detection system that combines data-driven training with expert knowledge to form a host-based intrusion detection system (HIDS). We demonstrate a software pipeline for efficiently generating and evaluating various BN classifier architectures for specific datasets and discuss explainability benefits thereof.

Keywords: Bayesian networks, cybersecurity, explainable machine learning, semi-supervised learning, discretization

1. Introduction

Prior work has used Bayesian networks (BNs) as the core technology of an intrusion detection system (IDS), albeit typically using a single BN architecture. Jemili et al. (2007) designed a network-based IDS (NIDS) that uses signature recognition matched with known behavior in combination with the K2 BN learning algorithm and Junction Tree inference. They focus on detection of intrusions but do not consider false positives, a very important metric in an IDS. Xu and Shelton (2010) present a system for both NIDS and HIDS based on continuous-time BNs which they employ in lieu of dynamic BNs due to the bursty nature of cyber event data. Their system focuses on event timing instead of complex features even though their HIDS data has somewhat imprecise timing which could allow incorrect event ordering. Jabbar et al. (2017) focus on increasing the detection rate and accuracy while attenuating the number of false alarms in an IDS using feature selection in combination with a BN classifier. Our approach seeks to achieve these same goals via more sophisticated techniques for feature selection and discretization and generates a suite of BN classifiers with performance tradeoffs.

Our focus on host-based cyberattack detection requires the use of host logs. Rather than the oft-used DARPA KDD '99 dataset (or its revised version, NSL-KDD), we utilized a more modern dataset from Sandia National Laboratories which contains a significant amount of host-based log data generated by Windows System Monitor (a.k.a. Sysmon) with millisecond timing, as described in detail in section 2.

BNs are probabilistic graphical models capable of multi-directional inference among multiple variables via Bayes' Rule (Pourret et al., 2008). When trained upon expert-informed features, BNs yield relatively interpretable solutions to classification problems. They are lightweight and cheap to train, natively provide confidence estimates and goodness-of-fit measures, and are relatively robust to imbalanced datasets and overfitting, especially in rare event detection (Uusitalo, 2007). These advantages make BNs well-suited to the detection and analysis of suspect system logs in the cybersecurity field. We have developed a pipeline that allows multiple BN architectures to be evaluated to determine the highest performing BN classifier solution for specific stakeholder needs with minimal manual effort.

We desired the capability to be accessible from a common data analytic environment. There was no Python-native comprehensive package for creation, training, and testing of BNs beyond Naïve Bayes, and although R has several strong BN packages, its copyleft licensing is unsuitable for some intended uses. After reviewing several BN modeling tools, we selected Bayes Server (2021) as our engine for BN creation, inference, and analysis, due to its Python-accessible API and strong data-driven model-building algorithms. We created Python wrappers for select components of Bayes Server's Java API, enabling model training and assessment within an automated workflow via a few succinct classes and methods.

We investigated several changes to manual BN development pipelines to improve convenience and classification performance. Key examples:

- Automatic discretization removes the need to manually discretize new/updated data and often outperforms continuous encodings; supervised discretization can further improve performance.

URI: https://hdl.handle.net/10125/102784
978-0-9981331-6-4
(CC BY-NC-ND 4.0)

- Automatic feature selection reduces model size and can improve classification performance.
- A performance-tunability metric and Pareto method enables selecting the best model at a given complexity.

We designed the framework for model interpretability and results explainability, as outlined in Figure 1:

- **Expert-informed datasets:** Start with datasets vetted by experts to ensure the pedigree and content of known malicious behavior (Section 2)
- **Interpretable features**: Derive features based on expert input, with names/descriptions which map to known suspect behavior. (Section 2)
- **Explainable discretization**: Define feature-split thresholds via a method explainable in terms of importance to target classification. (Section 3.2)
- **Bayesian Networks**: Provide interpretability by defining the relationships among variables (features and target/label) both graphically and with inspectable conditional probability tables.
- **Results explainability**: Enabled by BNs, provide explainability of output classifications via impact analysis, value of information, and analysis of difficult/ambiguous cases. (Sections 3.4, 5.3, 5.4)

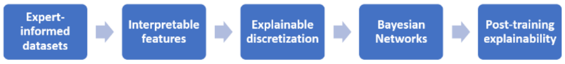


Figure 1. Visual outline of interpretable/explainable design pipeline.

2. Expert-Informed Features and Data

We conducted our analysis using two datasets, Tracer FIRE 9 (TF9) and Tracer FIRE 10 (TF10), which consist of a combination of both "normal", nonsuspicious system events, as well as "suspect" events resulting from adversarial attacks. The data was obtained from the Sandia National Laboratories (SNL) Tracer FIRE team. Tracer FIRE (TF) is a Forensic Incident Response Exercise designed by SNL to give participants an advanced persistent threat (APT) real-life scenario driven experience to test forensic skills and learn new methodologies to conduct forensic investigations. Each year a new scenario is created using the latest exposed APT attacks and software vulnerabilities used. Vulnerabilities are embedded into a simulated enterprise network with normal security practices in place, then attacked by a red team using customized APT malware. There can be multiple APT groups with differing tactics, techniques, and procedures (TTPs) using custom malware to gain access and perform actions necessary to their motivations. While each fictional APT performs its actions, forensic details are captured by the victim network using a variety of cyber event detectors including Zeek and Windows System Monitor (a.k.a. Sysmon). Access to the TF9 data is publicly available including the raw event data as well as the forensic reports which characterize the malicious behavior (Tracer FIRE, 2021).

To convert the raw Sysmon data into a usable form for training a BN, we developed a scenario extraction tool (SET) for identifying scenarios (parent-child process trees) which contain one or more suspicious events (system logs) based on expert-informed indicators and expert knowledge of cyberattacks. The scenarios labeled as "suspect" are known malicious APT activities within the dataset which may not readily map to intrusions identified in the MITRE ATT&CK matrix. The collection of suspect indicators and associated Sysmon events used herein are described in Table 1 and are intended to be illustrative but not exhaustive. The SET uses wildcard string matching to detect suspicious Sysmon events, as well as whitelisting to ignore innocuous events.

Each scenario is represented as a collection of aggregated statistics based on the events which comprise it as described below, then used as an interpretable set of features for the BN.

- *known company percent*: The integer percentage (0-100) of the executables within the scenario with a known company as the publisher. If the publisher is known, it is specified in the Sysmon event record, else it is blank.
- *file create count/duration/stdev*: The number of times the same executable repeats a file creation action; the duration in seconds; and standard deviation, when multiple file saves occur (three separate features).
- *max time delta*: The maximum time delta between any two adjacent scenario events (in seconds).
- *max tree depth*: The maximum depth of the process tree across all events in the scenario.
- *duration*: The sequence duration (starting from the first suspect event) in seconds.
- *threat [XX] count*: The number of times threat event *XX* from Table 1 was present in the scenario (e.g., Threat 17 count = # of file saves) (multiple separate features).
- *priority sum*: The sum of priority values across all child events, including repeats. The priority of an event indicates how suspicious it is (with higher values being more suspicious). The event prioritization scheme is based on expert input but used for notional purposes only.
- *max priority*: Maximum priority across all events.
- *single-dest count/duration/stdev*: The number of network connections from the same executable to a single destination (beaconing); the duration in seconds; and standard deviation when multiple network connections occur (three features).
- *multi-dest count/duration/stdev*: The number of times the same executable connects to multiple network destinations (reconnaissance); the duration in seconds; and standard deviation, when multiple connections occur (three features).

Table 1. Suspect indicators

[ID] Indicator Name	Associated Sysmon Event[a]	Priority
[1] WScript creating script in Users subdirectory	(11) FileCreate	2
[2] Execution of VB script in Users subdirectory	(1) Process creation	2
[3] PowerShell WebClient downloadstring	(1) Process creation	1
[4] PowerShell EncodedCommand	(1) Process creation	1
[5] wget storing exe file in Windows directory	(11) FileCreate	3
[6] wget creating exe file in any directory	(11) FileCreate	2
[7] MS Office creates exe file in Users subdirectory	(11) FileCreate	4
[8] MS Office creates any file in Users subdirectory	(11) FileCreate	2
[9] MS Office exe stream creation	(15) FileCreate-StreamHash	4
[10] Shell command launches exe	(1) Process creation	1
[11] Shell command launches exe in Users, Temp or Startup directory	(1) Process creation	2
[12] Executable launches power/command shell	(1) Process creation	1
[13] Any exe modifying registry	(13) Registry-Event (set)	1
[14] Exe in Users subdirectory or temp directory making network connection	(3) Network connection	3
[15] Exe in user Windows directory making network connection	(3) Network connection	1
[16] Exe in Users subdirectory creating DLL/EXE/script in Users, Temp or Startup directory	(11) FileCreate	3
[17] Browser saves DLL, EXE, or script in Users or Temp directory	(11) FileCreate	3
[18] Exe creates DLL/EXE/script in Users, Temp or Startup directory	(11) FileCreate	2
[19] Execution of exe in Users or Temp subdirectory	(1) Process creation	2
[20] Exe in non-main root directory creates exe in temp or downloads directory	(11) FileCreate	2
[21] Browser saves DLL, EXE, or script in users temp/downloads directory via FileStream	(15) FileCreate-StreamHash	3
[22] Shell command saves DLL, EXE, or script in Users, Temp or Startup directory	(11) FileCreate	2
[23] Suspect exe modification in registry	(13) Registry-Event (set)	3
[24] Suspect process creation	(1) Process creation	2
[27] Execution of exe in non-main root directory	(1) Process creation	2
[28] Execution of ping.exe (network discovery)	(1) Process creation	2
[29] Shell command launches script in Users, Temp or Startup directory	(1) Process creation	2

[a] a complete description of all Sysmon events can be found at https://docs.microsoft.com/enus/sysinternals/downloads/sysmon

3. Key Elements of BN Pipeline

The classification pipeline is based upon a Python API extension library written to provide a convenient level of abstraction for BN operations such as network instantiation, parameter and structural learning, batch queries, and explainability analyses. The library utilizes wrapper classes that succinctly encapsulate common BN operations from Bayes Server's Java API and facilitate integration with other machine learning libraries, allowing for creation, training, and inspection of BNs to be completed in a few high-level object-oriented function calls. The following subsections describe key pipeline features.

3.1. Feature Selection

Several schemes for feature selection were developed and tested with an eye towards improving data dimensionality and model interpretability and robustness. A basic scheme is to remove highly correlated features and features with zero variance, which we refer to as covariance screening. Additional feature-selection schemes using supervised learning algorithms are optionally applied after. Feature selection schemes utilizing mean decreases in impurity (MDIs) calculated by random forests, gain calculated by extreme gradient boosting (XGBoost) models, or ridge regression coefficients can often improve precision-recall performance of BNs.

3.2. BN Variable Discretization

Although Bayes Server's API (and therefore our library) supports continuous linear gaussian variables, their usage often requires significant additional BN structure complexity to adequately approximate joint distributions, and/or reduces model performance. Binning feature values and then using discrete variables is common, but it is expensive for a subject matter expert or analyst to analyze each variable separately to determine bin boundaries – particularly as datasets change and/or new features are engineered. Readily available programmatic discretization routines did not meet our needs. Thus, we investigated automated discretization schemes involving both supervised and unsupervised machine learning techniques. A primary challenge was to automate decisions that would normally require human input, such as number of bins, location of bin boundaries etc., in an explainable classification-salient way.

One solution was to run k-means clustering on individual features to discretize values into distinct clusters. The number of clusters to ultimately use for each variable was determined by the "elbow method" outlined by Bholowalia, et al. (2014); as the number of clusters was increased, the mean squared error (MSE) for each number of clusters was calculated and the optimal number of clusters chosen when the MSE dropped significantly, which we detected by approximating the second derivative with respect to the number of clusters using the central difference. Note that our goal was not to minimize MSE, but to provide a good balance between underfitting and overfitting. Double-interval partitioning values were then obtained by extracting k-mean centroids from the k-value that had the highest central difference. We refer to this approach of discretizing variables using k-means clustering as "k-means discretization".

A limitation of both the manual and k-means discretization methods is that they do not consider the relationship between the feature and the target variable; a bin boundary that "clusters" the feature well may not provide the best contrast for target classification. With this in mind, we modified the "random forests discretizer" proposed by Berrado and Runger (2009). Their method allows discretizing features in a supervised, multivariate manner that is relatively cheap to train: a random forest model is trained on the labeled data, and feature-split thresholds that are most important for classification can then be extracted by crawling through the forest.

However, as noted by Cheng (2015), raw count of a threshold's prevalence across all trees may be a misleading metric of importance. Building a weighted histogram based on change in Gini impurity at each split in every tree provides a more meaningful measure of the information content of a given feature split (see Figure 2). We therefore modified Berrado and Runger's algorithm using delta-Gini-weighted feature split importance histograms. Figure 2 illustrates this concept for feature "max tree depth"; note that splits below value 3.5, for example, provide low delta impurity and class contrast despite their prevalence. In Figure 3, Gini-based discretization of this same feature demonstrates high contrast between positive and negative classes.

Although Berrado and Runger suggest a possible thresholding method for future automation, the paper examples are based on manual selection of histogram modes. Needing a method to automatically determine the number of modes/bins to select and, recognizing that a choice of threshold is likely data-dependent, we instead developed an elbow method for automatically picking bin boundaries using the central difference, similar to the terminating method used for k-means discretization. Our modified random forest ("target-informed discretization") method is as follows:

1. Train a random forest classifier on the labeled training data and, for each feature variable, extract classification importance information for feature-split thresholds, as measured by changes in Gini impurity in every tree.
2. For each feature, build a weighted histogram with thresholds on the x-axis and split importance on the y-axis, with bin centers as candidate feature splits/thresholds.
3. Choose the number of feature splits for each variable using the elbow method over MSE scores. MSE for different numbers of splits are calculated and the optimal number of splits is determined by the central difference. Histogram bins the with highest relative importance are greedily added first, and histogram bin centers are used directly as our discretization bin boundaries (since they are chosen specifically for their ability to split the data for classification).

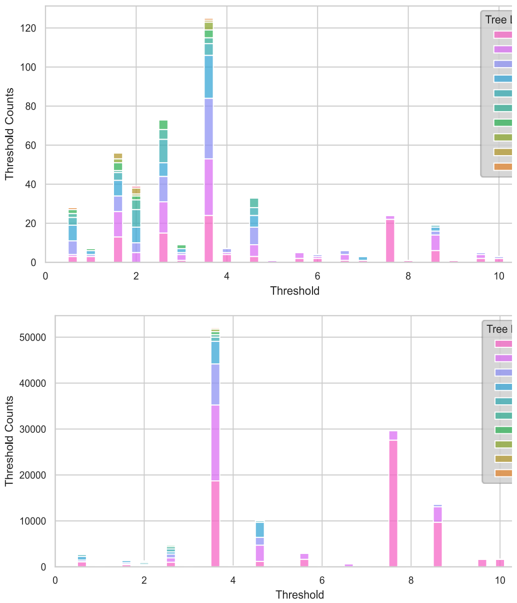


Figure 2. A comparison of measures of feature split importance for a numerical feature (max tree depth) in a random forest. The top plot shows raw count/prevalence of split values across all trees. The bottom plot shows counts weighted by changes in Gini impurities. In this example, we see that some split values with lower count (such as 7.5) provide greater delta impurity. In contrast, splits below 3.5 are quite frequent but provide low delta impurity and would add little value for classification.

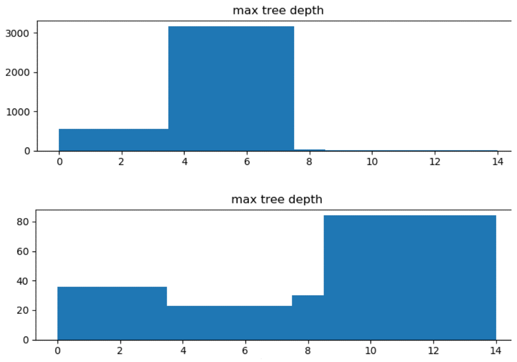


Figure 3. Histogram of max tree depth values, with bin boundaries set based on delta Gini impurities in Figure 2. Note contrast between negative (top) and positive class (bottom).

Key challenges included determining the number of bins/split candidates for the elbow method to work well, and how to decide whether/when a variable with a moderate number of distinct states should be further summarized in a discretized variable with fewer states. We further recognize that performance of this method is potentially hampered by the simplicity of the greedy heuristic and may be quite dependent on the binning strategy in step 2; we intend to address these questions more thoroughly in future work.

3.3. Assessing Models: Performance-Tunability Pareto Front

To assess BN performance, we took the geometric mean of the area under the Precision-Recall curve and the F1 score (at 0.5 probability threshold for positive classification). We will refer to this metric as the "performance-tunability" metric. The intent is to reward models that have good default performance *and* robustly tunable performance across alternate thresholds. Once multiple networks are trained and tested on the same data, the performance-tunability metric and number of parameters can be extracted from each BN and plotted. We propose selecting models that lie on the Pareto front of the plot, i.e., no other model can do better in one axis without getting worse in the other. An example is shown in Figure 7; since we wish to maximize model robustness and minimize model complexity, the Pareto front here is the upper left edge of the point cloud. Pareto front selection provides a means for selecting from a suite of models with optimal performance and complexity tradeoffs and can easily be adapted to other performance or complexity metrics.

3.4. Impact Analysis: Enhancing Classification Explainability

Impact analysis indicates which features make the greatest difference to the classification probability for a given set of evidence (Bayes Server, 2021). Evidence from a given data sample is included, one feature set at a time, in a query to a BN with no other evidence observed (or the evidence can be excluded from a query where all other evidence is observed). This capability leverages BNs' unique abilities for inference with missing feature data. Whereas manually setting evidence feature-by-feature in a UI is time-consuming and error prone, automating via the API allows the user to initiate a query directly from common Python data structures representing the entire feature space of a single sample/case (and, if desired, automate over all cases in a set/list).

We introduce a network graph visualization of impact analysis (executed for all individuals and pairs of features) that provides single-feature information within nodes and information for pairs of features along edges (see Figure 4). The results of such analyses can then be used to discern the impact of feature states or combinations of feature states against a particular target variable.

4. Experiment on Cybersecurity Data

We applied the pipeline capabilities above to the classification of cybersecurity scenarios (summarized process trees of system logs) found in the TF9 and TF10 datasets. Manually selecting the right BN classifier is a challenge; using the transformed TF9 and TF10 SET data, our analysis sought to determine which BN structures have the highest performance for scenario classification. We additionally sought to answer the following questions in the specific context of this application and dataset:

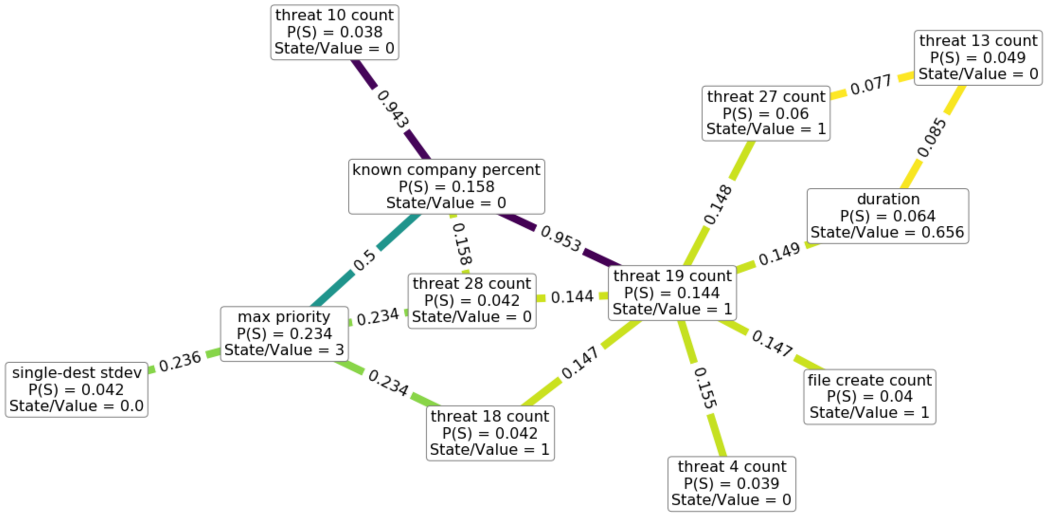


Figure 4. A graphical depiction of impact analysis queried on a BN trained on cybersecurity data, using feature state values from a single scenario. Nodes indicate feature names, states, and positive class probability if *only* this feature's evidence is observed. Edges indicate positive class probability if the two connected features' evidence is observed. The graph is filtered to show only combinations of evidence that substantially increase the probability of a positive classification (vs. no evidence). The darkest links belong to the subgraph of features most strongly tied to a positive classification. This model believes that the combination of 0 known company percent with 1 threat 19 count or 0 threat 10 count are the most suspect in the absence of other evidence. A similar graphic can be produced to understand which features/pairs are most impactful when *excluded* from the scenario evidence.

- Do BN classifiers benefit from semi-supervised training? Since BNs can learn non-target links from unlabeled data, can augmenting the training set with unlabeled data improve classification?
- How do feature selection and discretization – supervised methods in particular – affect the robustness of BNs?

Our experiment used a modification of k-fold cross-validation. The TF9 and TF10 data were aggregated into a single dataset and split into three separate train/test folds by shuffling and then stratifying across the overall dataset. We did not use more folds due to the highly imbalanced nature of the data (TF9 had 54 positives in 310 scenarios and TF10 had 119 positives in 3623 scenarios). We then defined three separate train-test partition strategies, described in Table 2:

Table 2. Partition strategies

Partition Strategy	Percentage of TF9+TF10 Data in Partition		
	Labeled train	*Unlabeled train*	*Test*
1-0-1	33.3%	0%	33.3%
1-1-1	33.3%	33.3%	33.3%
2-0-1	66.7%	0%	33.3%

The “2-0-1” strategy is equivalent to regular k-fold cross-validation with three folds and is therefore also referred to as a “2:1 train-test split”. The “1-0-1” and “1-1-1” partition strategies are included to measure the performance of semi-supervised training. For each strategy, the test set was rotated among the three folds.

Four preprocessing sequences were assessed by running k-means or Target-Informed discretization in combination with either Covariance Screening (0.99 threshold), or Covariance+XGBoost feature selection (0.99 cov. threshold, 0.001 XGBoost gain threshold).

Models were built via every individual structural learning algorithm shown as a node in Figure 5, and every valid pair shown as an edge (e.g., TAN followed by PC is valid, but PC followed by TAN is not). The full experiment design is summarized in Table 3.

Finally, parameter learning was performed on each model built from a unique sequence of train/test split partitioning, preprocessing, and structural learning, and metrics were averaged over the three test splits.

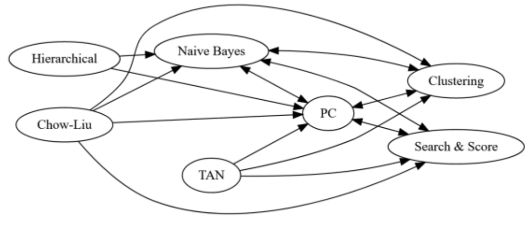


Figure 5. Nodes denote structural learning methods and edges denote valid ordered pairs (some methods do not support link constraints or latent variables from prior methods). Naïve Bayes is custom (to ensure an acyclic graph), and clustering is adapted to use the elbow method for fewer latent cluster states. All other methods are unmodified from Bayes Server.

Table 3. Experiment design

Train-test Partition Strategy		Pre-processing (on training data)				Structural Learning (on training data)
	×	*Screening*		*Discretization*	×	
1-1-1		Covariance	×	k-means		All methods
1-0-1		Covariance + XGBoost		Target-informed		All valid pairs of methods
2-0-1						

We assess each model’s performance via precision and recall (and related F1 and PR AUC measures), which are more appropriate for our class-imbalanced problem than TPR, FPR, accuracy, or ROC AUC. For simplicity, we omitted other imbalance mitigations, such as oversampling or overweighting the minority class, from the experiment design. Among machine learning (ML) methods, BNs are known to be relatively robust to imbalance (e.g., Leong, 2016) – likely in part because they are not typically trained to maximize accuracy – and in our experience over-weighting the minority class does not improve a BN’s classification performance. However, using SMOTE (Synthetic Minority Oversampling Technique) or other similar approaches before structural learning may result in better BNs from some algorithms. We intend to address class resampling and/or reweighting for structural learning in future work.

5. Cybersecurity Experimental Results

The results of the analysis are summarized in figures 6 and 7. In each figure, the top 20 models (by F1 score mean) from each preprocessing and partition combination are displayed, with each model being trained with a different sequence of one to two structural learning methods. Red and blue lines mark the performance of the best Naïve Bayes and tree-augmented naïve (TAN) models respectively, from any partition and processing strategy, with respect to the performance metric shown on the axis. The best model is not necessarily the same on all axes, so not all red/blue “crosshairs” will intersect a point. Values (other than the area calculation for performance-tunability) are calculated at the default 0.5 probability threshold for positive classification.

The efficiency and comprehensive feature set of the Python API extension library allowed for the creation of highly robust models for this specific application. Given a rich training set and adequate discretization and data preprocessing, we saw mean F1 scores surpass 0.87 in a highly data-imbalanced classification application (Figure 6). Through combining structural learning methods, models often exceed the performance achieved by traditional Naïve Bayes and TAN models within the same partition and pre-processing classes (recall that in Figure 6, only the *best* TAN and Naïve Bayes models across *all* strategies are denoted by blue and red lines).

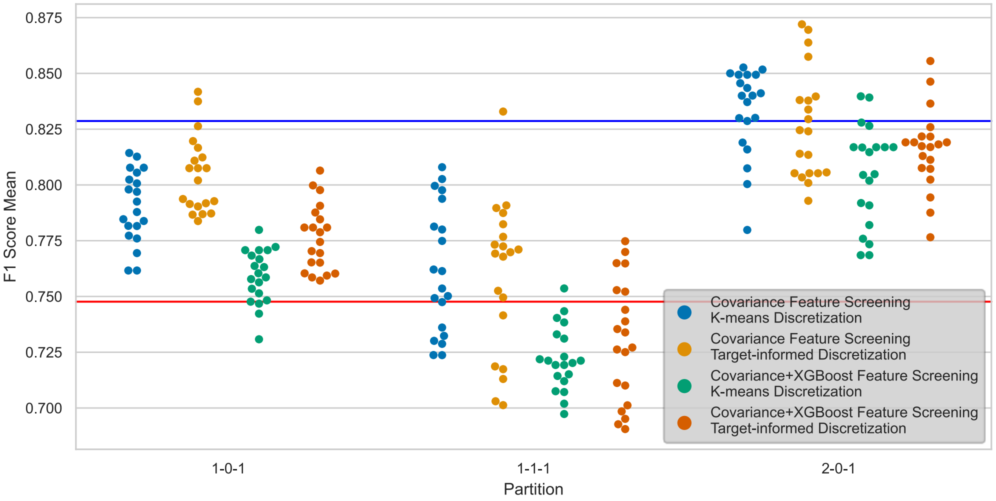


Figure 6. Mean F1 score across 3 test splits of best 20 models for each preprocessing strategy (colors) and partition strategy (x-axis). Red line = F1 of best Naïve Bayes, Blue line = F1 of best TAN.

The best F1 performance was from models using the 2-0-1 train-test strategy (unsurprisingly) and built using covariance screening and target-informed discretization. Models with high F1 are generally also tunable, based on correlation between mean F1 and mean area under the precision-recall curve ($R^2 = 0.71$). This indicates that competitive models can likely be tuned to specific probability thresholds to achieve greater performance in either precision or recall. BNs also performed quite well with sparser labeled training data (1-0-1 section of Figure 6), particularly using target-informed discretization and the less aggressive covariance screening. Going from 1-0-1 to 1-1-1, adding unlabeled data *decreased* mean F1 scores. This is consistent with Cohen et al. (2003), which observes that addition of unlabeled data to a training set can often reduce performance on model structures not designed to take advantage of it.

5.1. Pareto Front Selection

Plotting performance-tunability versus parameter count in Figure 7 as discussed in section 3.3, the Pareto front (top left) selects models with simple underlying structures to optimally balance complexity and performance – typically built with naïve, TAN, or clustering structural learning. K-means discretization generally provides lower complexity, particularly combined with XGBoost feature selection. Higher performance often results from covariance screening and/or target-informed discretization. Notably, the choice of axes for the Pareto could differ based on stakeholder priorities. In the cybersecurity domain, analysts are typically swamped with false positives, so choosing models with high precision and acceptable recall is a likely strategy. We could instead select the Pareto set of the mean precision vs. mean recall plot to

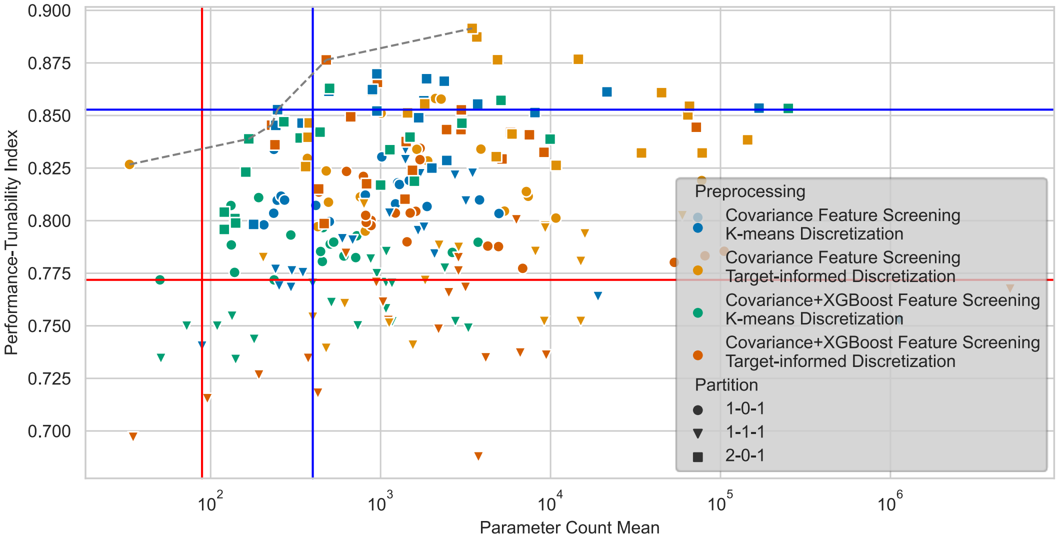


Figure 7. Mean performance-tunability index vs. mean parameter count across all 3 splits.

get optimal precision-recall tradeoffs. This strategy favors search-and-score and naïve/TAN structures, often in combination with other methods, and favors the 2:1 train-test strategy and covariance screening.

5.2. Comparison Against Random Forest

Table 4 compares metric averages of the best-average-F1 model – naïve and clustering BN (NBC) – for the 2-0-1 split against default random forest and TAN BN models. This single-BN model developed by our pipeline performs quite favorably when compared to common naïve and TAN structures and remains competitive when compared to random forest models in terms of F1 scores. Even greater F1 scores were possible on our testing set by running a grid search over different probability thresholds.

Table 4. Classifier Performance Comparison

	Random Forest	TAN (PT=0.5)	NBC (PT=0.5)	NBC (PT=0.13)
Accuracy	99.3%	98.4%	99.0%	99.0%
TPR/Recall	85.0%	80.9%	81.6%	89.0%
FPR	0.05%	0.82%	0.24%	0.56%
Precision	98.0%	82.0%	94.1%	88.0%
F1 Score	91.0%	81.4%	87.2%	88.5%

The slightly worse NBC performance compared to a random forest was deemed acceptable considering the vastly greater interpretability of a single BN with inspectable probability tables vs. a large forest of decision trees, and the local/global self-explainability it provides for subject matter experts (although random forests have built-in *global* explainability, they require a surrogate like SHAP to derive local explainability for specific samples). Specific interpretability and explainability benefits of BNs are discussed further in sections 5.3 and 5.4.

5.3. Interpretability and Explainability

As pointed out by Rudin (2019), it is preferable to use inherently interpretable models for making high-stakes decisions; cyber intrusion detection certainly qualifies. Rudin further claims that any explanation method for a black-box model will almost certainly be inaccurate for some inputs, resulting in unpredictable mismatches between the model and explanations. BNs avoid these issues by providing an interpretable model representation as well as self-explanation of output (i.e., without needing a surrogate).

Figure 8 shows a TAN BN model trained on the Tracer FIRE data with highly correlated variables (34% of features) removed; it illustrates the dependencies between the different variables for model interpretability. The conditional probability table (CPT) for each variable can be inspected to get a deeper understanding of the classifier.

Aas et al. (2020) mention that a disadvantage of calculating Shapley values – as required to explain many common machine learning methods – is that the computational complexity grows exponentially with number of features, which has led to approximation methods being used by the Kernel SHAP method. The Kernel SHAP method assumes that the features are independent and when this is not true (i.e., when some variables are correlated) may lead to incorrect Shapley values and thus incorrect explanations. Further noted by Smith et al. (2021), LIME and SHAP make strong assumptions of feature independence and linear interactions, which are frequently inaccurate. In contrast, not only are BN models self-explainable, but because they are generative models, explanation does not require additional calculations over the training set. Thus, no subsampling is needed, and explanation accuracy does not vary by sample size. Furthermore, explanation calculation runtime is independent of training set size and can be dramatically faster than post-hoc explanation methods.

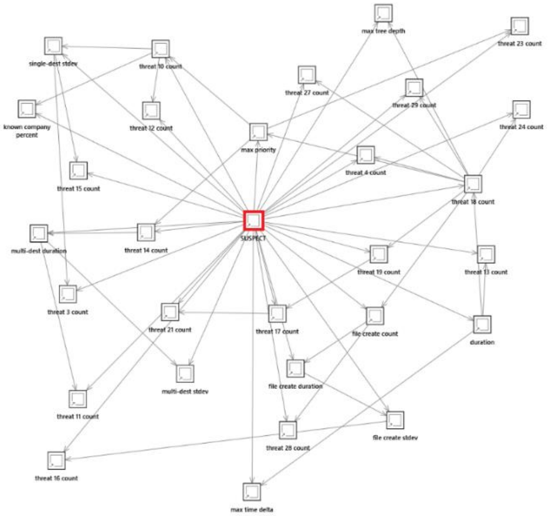
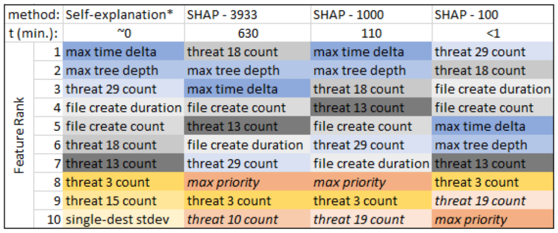


Figure 8. TAN BN trained on TF9 and TF10 data. Target variable SUSPECT is highlighted in red.

To assess SHAP explainability versus BN self-explainability, we applied the Python implementation of Kernel SHAP (Lundberg & Lee, 2017) to a Categorical Naïve Bayes classifier in scikit-learn (CategoricalNB [Pedregosa et al., 2011]) using different sample sizes for the SHAP analysis: all 3933, 1000 and 100 samples. The experiments were performed on a Windows 10 PC with a dual-core i7-7500U running at 2.70 GHz. We saw exceedingly long run times (especially when using all 3933 sample records) and an inconsistency of feature-importance ordering between runs for sampled SHAP. As shown in Table 5, the Kernel SHAP feature importance does not match well with the feature importance given by the Value of Information of the Categorical Naïve Bayes model (as produced by Bayes Server with an equivalent model). Given this mismatch between the BN's self-explained feature importance (which comes directly from the BN) and the SHAP determination of feature importance, one could reasonably hypothesize that using SHAP as a post-hoc explainability method can result in misleading or incorrect feature importance explanations for other ML models as well.

Table 5. Runtime and top-10 features for Naïve BN self-explanation vs. SHAP



method:		Self-explanation*	SHAP - 3933	SHAP - 1000	SHAP - 100
t (min.):		~0	630	110	<1
Feature Rank	1	max time delta	threat 18 count	max time delta	threat 29 count
	2	max tree depth	max tree depth	max tree depth	threat 18 count
	3	threat 29 count	max time delta	threat 18 count	file create duration
	4	file create duration	file create count	threat 13 count	file create count
	5	file create count	threat 13 count	file create count	max time delta
	6	threat 18 count	file create duration	threat 29 count	max tree depth
	7	threat 13 count	threat 29 count	file create duration	threat 13 count
	8	threat 3 count	*max priority*	*max priority*	threat 3 count
	9	threat 15 count	threat 3 count	threat 3 count	*threat 19 count*
	10	single-dest stdev	*threat 10 count*	*threat 19 count*	*max priority*

*as measured by mutual information with, or hypothesis entropy reduction for, the SUSPECT variable

Because the BN is a collection of CPTs, it is also relatively straightforward to determine via Bayes Rule how the target variable relates to individual *states* of the features. This characteristic enables understanding the effects of dependent/ correlated features and nonlinear interactions and can provide valuable feature insights. For instance, a state-level analysis of the Naïve Bayes model shows that low *max tree depth* values provide little target contrast, and virtually all of the discriminative power of the feature lies in the difference between moderate and high values. Such analysis can also be performed with partial evidence observed, allowing the user to understand what feature state information would be next-most-valuable when some features are known.

5.4. Analysis of Difficult/Ambiguous Cases

A given BN classifier model can be analyzed to determine which cases it found the most difficult to score. By comparing the log likelihood of each case with and without the target label observed, one can find which cases would have significantly better model fit without the assigned label. This analysis process allows identifying potentially mislabeled cases and cases whose label-salient characteristics are perhaps not well represented in the feature space.

We demonstrate here on a model with excellent precision and moderate recall (chosen consistent with a priority on minimizing false negatives). As seen in Figure 9, there are a few cases with large differences between the log likelihood scores with and without target labels observed. As one would expect given the model's high precision and modest recall, all of these cases were false negatives in at least one test split. We discuss three representative cases:

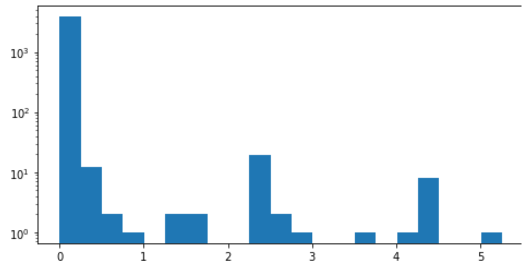


Figure 9. Histogram of labeled minus unlabeled log likelihood across all TF9 and TF10 cases (average across 3 test splits) using a high-precision BN model. Note log-y axis.

- TF9-240: File downloads followed by sequence of chained command/powershell executions – suspicious upon manual inspection, but in ways that are not well reflected in current features.
- TF10-118: File downloads, apparently innocuous registry sets, and download of a PowerShell script. Weak evidence of malicious activity without prior knowledge.
- TF10-1078: Save of a batch file by svchost.exe. Weak evidence of malicious activity without prior knowledge.

In these cases, it is understandable why the model provided false negatives (and why models that label these correctly might have worse precision). Such cases are expected, since APTs intentionally perform attacks using standard techniques to mimic normal host behavior and avoid detection. Additionally, cases such as TF9-240 may motivate iterative refinement of the feature space when existing features do not reflect key nuances relevant to the classification problem.

Ambiguous cases can also be identified based on classification probability, and impact analysis (as discussed in section 3) may help clarify the model's labeling rationale and give analysts more evidence to determine whether a classification might be incorrect. Figure 4 depicts such a case, TF9-122, which was highly ambiguous (P(Suspect) = 0.4996) in one test split. Review of an impact analysis visualization such as Figure 4 would allow an expert to determine the salient features of such a case and determine whether the evidence was sufficient for a given classification.

6. Conclusions

This study demonstrated the effectiveness and efficiency of building BNs for detecting cyberattacks through automated pipelines using a novel Python API extension library. Automating structural and parameter learning, discretization, and k-fold cross-validation accelerate model building and assessment and enable more comprehensive experimentation and model optimization. In particular, the ability to automate structural learning across a wide selection of algorithm combinations enables creation of a diverse portfolio of networks, improving the likelihood of creating appropriate models for a wide range of use cases and stakeholder needs. As APT behavior drifts, models can quickly be rebuilt or retrained.

Using this pipeline, we can efficiently create a cyberattack detection system which considers the tradeoff between the probability of detection and the probability of false alarm. This tradeoff can be considered both during model selection, and in tuning the selected model's detection threshold. Given the typically unbalanced data that will be experienced in real-world systems, managing this tradeoff is critical since missing a true attack could result in a significant compromise whereas excessive false positives make it far more difficult to single out an actual attack.

Calculating log likelihood of each case with and without the target label observed allows identifying cases that the model had difficulty classifying. This can lead to greater understanding of false negative/ false positive trends, and of where features can be improved. Batch impact efficiently provides tabular and/or graphical classification explanations for any desired data samples. This capability could aid cyber analysts in quickly determining if certain ambiguous scenarios are malicious or innocuous and why.

While other methods such as random forests may have greater classification performance in certain instances, the BN pipeline provides a suite of models that are mostly simpler and more interpretable, with built-in explainability (such as value of information, impact analysis, and analysis of difficult cases). In this application, because we have developed interpretable features, the BN variables (ignoring latent variables) map directly to the input data and there is no "semantic gap between real-world interpretation and low-level feature space" (Smith et al. 2021) as seen in black-box ML systems. These interpretability and explainability characteristics of BNs are critical to analyst and stakeholder trust in results as well as overall utility of the system.

7. Acknowledgment

This article has been authored by employees of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employees own all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan. SAND2022-12653 C

8. References

[1] Aas, K., Jullum, M., & Løland, A. (2020, February 6). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. arXiv.org. Retrieved May 25, 2022, from https://arxiv.org/abs/1903.10464

[2] Bayes Server (2021). Bayes Server (Version 9.5) [Bayesian Network & Causal Artificial Intelligence software]. Retrieved April 15, 2021, https://www.bayesserver.com/

[3] Berrado, A., & Runger, G. C. (2009). Supervised multivariate discretization in mixed data with random forests. *2009 IEEE/ACS International Conference on Computer Systems and Applications*. https://doi.org/10.1109/aiccsa.2009.5069327

[4] Bholowalia, P., & Kumar, A. (2014). EBK-Means: A Clustering Technique based on Elbow Method and K-Means in WSN. *International Journal of Computer Applications*, *105*(9), 17–24.

[5] Cheng, S. (2015, October 1). *Unboxing the random forest classifier: The threshold distributions*. Medium. Retrieved May 3, 2022, from https://medium.com/airbnb-engineering/unboxing-the-random-forest-classifier-the-threshold-distributions-22ea2bb58ea6

[6] Cohen, I., Sebe, N., Gozman, F. G., Cirelo, M. C., & Huang, T. S. (2003). Learning bayesian network classifiers for facial expression recognition both labeled and unlabeled data. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* https://doi.org/10.1109/cvpr.2003.1211408

[7] Jabbar, M. A., Aluvalu, R., & Satyanarayana Reddy, S. S. (2017). Intrusion detection system using bayesian network and feature subset selection. *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 1–5. https://doi.org/10.1109/iccic.2017.8524381

[8] Jemili, F., Zaghdoud, M., & Ahmed, M. (2007). A framework for an adaptive intrusion detection system using Bayesian Network. *2007 IEEE Intelligence and Security Informatics*, 66–70. https://doi.org/10.1109/isi.2007.379535

[9] Leong, C. K. (2016). Credit risk scoring with bayesian network models. Computational Economics, 47(3), 423-446.

[10] Lundberg, S., & Lee, S.-I. (2017, November 25). *A unified approach to interpreting model predictions*. arXiv.org. Retrieved May 26, 2022, from https://arxiv.org/abs/1705.07874

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011, February 1). *Scikit-Learn: Machine learning in Python*. The Journal of Machine Learning Research. Retrieved May 26, 2022, from https://dl.acm.org/doi/10.5555/1953048.2078195

[12] Pourret, O., Naïm P., & Marcot, B. (2008). *Bayesian networks: A Practical Guide to Applications*. Wiley & Sons.

[13] Rudin, C. (2019, September 22). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. arXiv.org. Retrieved May 26, 2022, from https://arxiv.org/abs/1811.10154

[14] Smith, M., Acquesta, E., Ames, A., Carey, A., Cuellar, C., Field, R., Maxfield, T., Mitchell, S., Morris, E., Moss, B., Nyre-Yu, M., Rushdi, A., Stites, M., Smutz, C., & Zhou, X. (2021, September 1). SAGE Intrusion Detection System: Sensitivity Analysis Guided Explainability for Machine Learning. (Technical Report) | OSTI.GOV. Retrieved May 31, 2022, from https://www.osti.gov/biblio/1820253

[15] Tracer FIRE (2021). Tracer FIRE 9 event data and reports, SAND2021-5906. Sandia National Laboratories. https://tracerfire9.s3.amazonaws.com/ossec_data.zip, https://tracerfire9-release.s3.amazonaws.com/TF9+Forensic+Reports.7z.

[16] Uusitalo, L. (2007). Advantages and challenges of Bayesian networks in environmental modelling. *Ecological Modelling*, *203*(3-4), 312–318. https://doi.org/10.1016/j.ecolmodel.2006.11.033

[17] Xu, J., & Shelton, C. R. (2010). Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research*, *39*, 745–774. https://doi.org/10.1613/jair.3050