Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

A practical approach for applying Machine Learning in the detection and classification of network devices used in building management

Maroun Touma*[1] | Shalisha Witherspoon[1] | Shonda Witherspoon[1] | Isabelle Crawford-Eng[2]

[1]IBM Research, NY, USA
[2]University of Pennsylvania, PA, USA

Correspondence
*Maroun Touma, Email: touma@us.ibm.com

Present Address
IBM Thomas J Watson Research Center 1101 Kitchawan Rd, Yorktown Heights, NY 10598

Abstract

With the increasing deployment of smart buildings and infrastructure, Supervisory Control and Data Acquisition (SCADA) devices and the underlying IT network have become essential elements for the proper operations of these highly complex systems. Of course, with the increase in automation and the proliferation of SCADA devices, a corresponding increase in surface area of attack on critical infrastructure has increased. Understanding device behaviors in terms of known and understood or potentially qualified activities versus unknown and potentially nefarious activities in near-real time is a key component of any security solution. In this paper, we investigate the challenges with building robust machine learning models to identify unknowns purely from network traffic both inside and outside firewalls, starting with missing or inconsistent labels across sites, feature engineering and learning, temporal dependencies and analysis, and training data quality (including small sample sizes) for both shallow and deep learning methods. To demonstrate these challenges and the capabilities we have developed, we focus on Building Automation and Control networks (BACnet) from a private commercial building system. Our results show that ”Model Zoo” built from binary classifiers based on each device or behavior combined with an ensemble classifier integrating information from all classifiers provides a reliable methodology to identify unknown devices as well as determining specific known devices when the device type is in the training set. The capability of the Model Zoo framework is shown to be directly linked to feature engineering and learning, and the dependency of the feature selection varies depending on both the binary and ensemble classifiers as well.

KEYWORDS:
BACnet; Model Zoo; Binary Classifier; ensemble

1 | INTRODUCTION

Securing building IT infrastructure is an ongoing concern for all building operations managers. One of the first steps in achieving a high level of resiliency against malicious attacks is to be able to account for all Operation Technology (OT) devices that are connected to the building network. In one manufacturing location, using our framework, we found over a third of the SCADA devices that are connected to the network are missing from the assets list provided by the building network administrator. In another scenario working with network data from a private lab, we find that a small but proportionately significant number of devices to be mislabeled as the result of human omissions or simply because some of the Internet Protocol (IP) addresses available for the site get re-assigned.

While many industry players and academic research advocate for implementing rigorous processes and tools to prevent these situations from occurring, in our research we demonstrate how passive inspection of the network traffic and the use of machine learning (ML) models can assist the human operator in correcting mislabelled devices and identify endpoints that are connected to the network even when those endpoints are not properly represented in the building assets database.

As Building Management Systems (BMS) become more complex with greater emphasis on automation and safety, the behavior of the devices that make up the solution changes as the BMS solution evolves to address new requirements. Such behavior is primarily defined by their communication patterns on the network. This includes the network protocols used, other endpoints in the network they communicate with, as well as the volume and frequency of the traffic sent and received. As the device communication patterns change, we are able to detect such change on a continuous basis using novelty detection for known/unknown techniques. This in turn triggers training of a new model that is added to the Model Zoo to detect the new behavior. While this approach is highly successful in tracking changes in the network, it is also highly susceptible to the quality of the data used for training our models. In this case, we will show how timeline analysis of device activities can be used to assess the quality of the data used for training.

1.1 | Identifying Behaviors and Unknowns: The Challenge

We characterize endpoint behavior based on three key aspects including 1) The communication protocols that the endpoint uses during the observation period, 2) The destination endpoints they communicate with and the nature of traffic and payload exchanged between the source and destination, and 3) the timeline for their communication activities organized as a series of fixed duration epochs.

In this paper, we limit our discussion to those devices that implement the BACnet/IP [1] communication protocol as it is commonly used in building automation. Such devices include sensors, actuators, switches, user terminals, and Programmable Logic Controllers (PLC) used in a diverse set of applications for backup power management, heating and cooling, etc. We exclusively rely on packet captures (PCAP) using a network tap that is passively observing all communications within the studied network. Training of the various ML models is done using features that are specific to each device using a one-class classifier (OCC), a binary classifier that is trained using in-class features only, and without any prior knowledge or assumptions of out-of-class features.

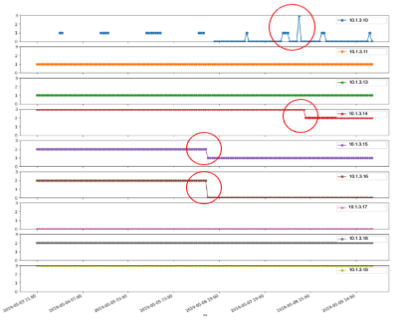


FIGURE 1 Tracking device cluster migration over a period of several days

A more traditional approach using Multi-class classifiers show reduced accuracy of the models when the observed device changes behavior over time. This change in behavior is commonly observed with SCADA devices. Figure 1 highlights this point by tracking device cluster migration over several days based on a one-hour epoch. While most devices remain stable in their own respective clusters, we observe that device 10.1.3.10 often migrates between clusters 0 and 1 and, at least in one instance, it migrates from cluster 0 to cluster 3. Similarly, device 10.1.3.14 migrates from cluster 3 to cluster 2 toward the end of the observation periods while devices 10.1.2.15 and 10.1.3.16 migrate from cluster 2 to cluster 1 and from cluster 3 to cluster 0 respectively almost simultaneously half-way through the observation period. We further demonstrate how this method has proven to work quite effectively for unbalanced training data sets since different devices generate a disproportionate amount of network traffic. Network taps generate a large amount of unlabeled data. This presents added complexity for training and assessing the performance of ML models. Partial and often incomplete labeling is commonly done by a network administrator using the IP address of the devices on the network to list their common attributes such as site-assigned identifier, device manufacturer, device type and function, and other attributes that are often site-specific. This approach is error prone and often uses confusing terms that are only meaningful to the network administrator of the local site. Furthermore, the spreadsheet that lists the devices and their attributes is often out of date and only updated when new devices are added and almost never when older devices are reconfigured or retired. Our research shows how using natural language processing techniques based on a domain specific corpus and information obtained from the analysis of the PCAP itself can provide an accurate identification and labeling of the devices on the network.

A key objective of our research is to demonstrate how ML can be used to learn and encode within the trained model the complex behavior of each BACnet device that was operating on the network during the observation period without any prior knowledge or description of the observed device or the environment in which it is operating. This is in clear contrasts to some of the techniques commonly used in the industry using firewall-like rules [2,3] and first order logic to identify each device on the network and control its activities based on its IP address.

ML models are notoriously sensitive to the quality of the training data. For network analysis, this is particularly sensitive to the procedure and techniques used to capture the PCAP. Using an empirical method for data quality analysis, we observe that human errors may account for the majority of bad quality in the data as the technical procedure for setting up and running the tap is relatively complex. Other causes also include the expected variability of the environment where the devices are operating, leading to a high degree of variability in the data, and making the task for selecting a stable set for training highly unlikely.

1.2 | Previous Work

Earlier works have presented a variety of methods for the temporal analysis of SCADA network data. Lin et al[4] for example created a methodology to extract traffic patterns from both physical lab experiments as well as an emulated electrical system. This methodology was then further used to describe device behavior and discover spontaneous events in the traffic flow. Our work instead uses temporal analysis of network traffic to characterize traffic patterns for feature engineering and identify issues with data set quality, such as temporal gaps in the data stream as well as the great diversity in data when captured over long periods of time.

Other previous works have also conducted research for behavior analysis in SCADA networks. Some works such as [5] have attempted to identify and leverage correlations between temporal based patterns in SCADA data and certain network behaviors for intrusion detection, whereas others like [6] have instead implemented certain shallow learning techniques, namely a One-class Support Vector Machine (OCSVM), as an intrusion detection mechanism for SCADA networks using packet rate and size features. Our methodology combines the use of both temporal analysis and shallow learning techniques similar to those employed in these works. In our case however, temporal analysis is specifically used for data quality assessment, and features used to train our chosen binary classifiers leverage additional information gleaned from the specific SCADA protocols chosen beyond simply packet rate and size.

More recently, researchers have additionally used deep learning techniques for feature learning and network behavior classification. For example in [7], a combination of a long-term, short-term (LSTM) neural network architecture and a stacked autoencoder architecture was used to learn features of IoT devices in an unsupervised setting. Additional deep learning techniques were then used to tune a clustering algorithm for the separation of learned features into distinct classes for later classification.

Other works have further explored the implementation of similar deep learning techniques for both feature learning as well as behavior analysis. In [8], a convolution neural network was instead used to learn features for classification in an attached softmax layer. Though similar deep learning techniques were also used in our research for feature learning, our work included a thorough investigation of learning features using encodings and embeddings as well. Moreover, in our work, probabilistic classifiers were used in place of deterministic networks for behavior analysis.

Finally, there has been prior art in label correction, which has included explorations in active learning of label corrections[9], as well as clustering-based methods, which groups instances together to infer the ground-truth labels[10]. However, the active learning approach assumes that samples with incorrect labels are known in the data set, which is not always known in advance, whereas the cluster-based approach assumes availability of a clean labeled subset for use as a standard. Additionally, label correction for BMS data pose their own specific issues, due to the fact that they are unstructured and must often be manually normalized to follow a consistent structure across different data sets[11]. In our work, we only assume a corpus that can be used to train our model to learn the relationship between labels in the training set, which can then be applied to our data set to clean labels, and make suggested corrections to the appropriate row/column. We also address inconsistency and manual normalization of BMS labels by seeking to resolve similar labels as the same entity using word embedding algorithms.

2 | METHODOLOGY

Analyzing network traffic presents a particular challenge as it relates to the volume and diversity of the data being captured. The first can be easily addressed using a datalake architecture that uses global namespaces to organize and index the data based on the network tap location, the protocol being used, and the day and hour of the data capture. Using a protocol parser, such as Tshark or Bro/Zeek, the captured PCAP stream is parsed into individual json records, each representing a single packet with a specific timestamp, source, destination, header, payload and transport layer protocol identifier (i.e. TCP, UDP, etc.). The payload is further broken out into one or more json records representing the specific elements associated with an application layer protocol (HTTP, DNS, BACnet/IP, etc.) and extracted from the payload. All json records generated using the protocol parser are then stored in parquet files that are organized using subfolders representing the various indices.

Feature vectors for training and scoring are retrieved from the datalake using pyspark aggregate functions based on a one hour epoch. Built into each query is a categorical encoding scheme for all nominal values of the protocol such as port numbers, service code, object types, etc. Feature vectors from multiple consecutive epochs are grouped together to form the bulk of the training data, and a randomized train/test split is used to train the various models and assess each model's performance. Feature vectors associated with each IP address in the training data are used for training a different OCC using in-class features only and producing a binary score with 'known' or 'unknown' labels as it relates to each IP address. As such, an IP network with N devices will lead to a Model Zoo with N different classifiers and one ensemble method[12]. A confusion matrix is then applied to reduce the number of classifiers when any one device consistently achieves a high score against another device model, indicating a high degree of similarity between the two devices. Using a second stage boosting algorithm, the scores from each OCC are used in an ensemble method to produce a 'known' or 'unknown' label as it relates to the whole IP network. Categorical encoding, summarization, and aggregate functions play a key role in engineering new features that are used by the Model Zoo. Raw features found in PCAP files are inherently noisy and the use of statistical analysis forms the basis for removing noise from the training data used for building new ML models.

2.1 | BACnet Protocol Features: A Brief Review

BACnet is a data communication protocol used widely for the control and automation of buildings. BACnet/IP is a UDP protocol that uses a client/server paradigm, in which a client device sends requests for specific services throughout the network, and other devices known as servers complete the requested service and in turn reply to the client with the results. BACnet exchanges are broadly classified as Network Protocol Data Unit (NPDU) or Application Protocol Data Unit (APDU). APDU messages includes Confirmed Service code (e.g read property, write property, etc.) where a reply typically containing data is expected by the client originating the request, and Unconfirmed Service code (e.g. Who is, I Am, etc.) where no reply is expected by the client[13]. All messages have a specific type (request, reply, alarm, etc.) and commonly reference objects that each device manages and their associated Object Types (Analog Input, Binary Output, command, etc.). Other attributes associated with the device such as a vendor identifier (e.g. Johnson Controls, Schneider Electric, etc) can also be found in specific replies. In this paper, we limit our discussion to the experimental results that are based on features that are derived from *Confirmed Services* and *Object Types* found in various BACnet-APDU messages.

2.2 | Temporal Considerations

Temporal analysis can be used in many ways to understand network behavior as a sequence of events. Methods such as k-shape clustering[14], [15] and graphical event modeling[16] can be used to understand general network behavior patterns over time. In our work however, temporal analysis was used largely for the assessment of network data quality for the training of both our shallow and deep learning models. We use a data set collected over a period of 6 days from a network configuration that includes 10

BACnet devices. Only *Confirmed Service* requests were taken into consideration when conducting the temporal analysis of the network data. A heat map depicting the relationship between a device and its number of requests made during the observation period is shown in Figure 2.

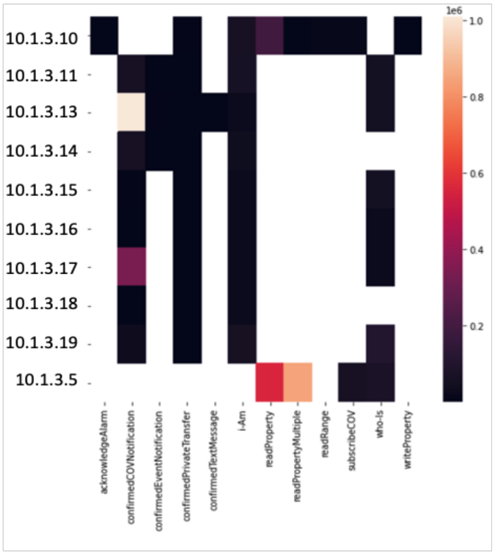


FIGURE 2 Heat Map of BACnet devices service requests

A timeline of all *Confirmed Service* requests made by all devices was generated. Figure 3 shows a typical hourly traffic pattern for the most active device. We observe that while some service requests are more periodic in nature than others, the general patterns for the device behavior can be identified using the time series for all its associated requests.

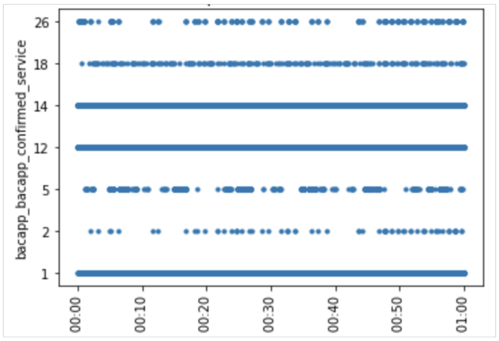


FIGURE 3 Timeline of Confirmed Service requests for one hour of network activity

Further temporal analysis of the network activity for the entirety of the day was additionally used to establish a typical traffic pattern for the network at a daily epoch. Figure 4 shows the timeline for all *Confirmed Service* requests made throughout the chosen day. While a handful of spontaneous rare events, including alarm services, object access services, and remote device management services, were also observed during this time period, network behavior at this time was largely consistent in nature with no significant data gaps.

This standard for network activity at both hourly and daily epochs was then used as a measure for data quality comparison for the remaining network data in the data set in order to identify any abnormal traffic patterns. Such an anomaly was observed in the

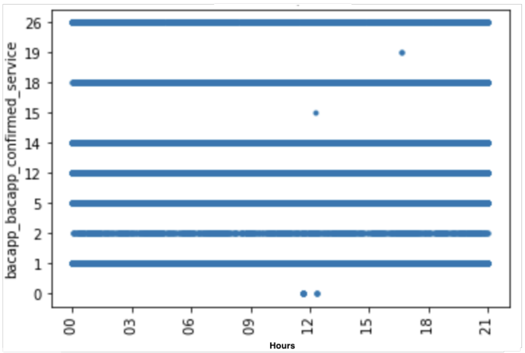


FIGURE 4 Timeline of Confirmed Service requests for one normal Day of network activity

network behavior for one day in a different month as shown below in Figure 5. On this day, though the types of *Confirmed Service* requests made by devices were similar to those observed previously, in stark contrast to the normal network traffic patterns, a significant three hour data gap was observed between the hours of 15:00 and 18:00. During this time period, no requests were made by any device on the network.

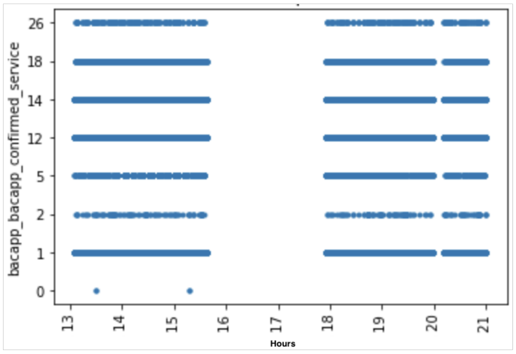


FIGURE 5 Timeline of Confirmed Service requests showing temporal anomalies for network activity

The establishment of a standard for network traffic patterns and behavior and comparison of other network activity timelines to such a standard helps to ensure that poor quality data with temporal gaps is not used for the training of the model. Temporal analysis additionally allows for the diversity of the overall data set and sparsity of particular rare events to be examined, thus allowing for difficulties faced by the model to be better understood.

2.3 | Label Generation and Completeness

One of the main challenges with labeling building data is the "manual effort currently required to normalize existing, heterogeneous building descriptions to a given standard" along with the fact that these labels are usually unstructured, building-specific, inconsistent, and reliant upon vendor-specific conventions for consistent interpretation[11]. Even with open-source initiatives like Project Haystack[17] providing metadata and labels for building data, problems still exist for adopting these efforts as a standard due to the same issues listed above. Due to the manual effort required to prepare a data set consisting of building data, it is likely that other issues in labeling arise that are non-exclusive to building labels, such as imputing blank values or mislabeling values in incorrect columns, which need to be identified and remedied by a data analyst, often in an iterative process. We addressed these issues by creating a semantic model for learning the relationship between labels in our data set, and creating a word-embedding model for which we can query to identify and address labeling issues previously stated.

2.3.1 | Generating Word Embedding Model

In order to learn the relationship between labels in our data set, we took an unconventional approach by utilizing techniques typical in natural language processing (NLP), such as creating word embeddings to represent text, which results in words with the same meaning having a similar vector representation, and using an alternate tabular data set consisting of SCADA devices and its attributes such as the device manufacturer, device type, function, and operating system, as our training set. Although our training set is not structured in the manner of an English sentence, we treated each singular column value as a word in our sentence, which would essentially have the same effect of learning which labels or column values typically appear together in the same context. Table 1 for example, shows a row from the corpus we used to create our word embeddings.

TABLE 1 Subset of corpus

device_manufacturer	**device_type**	**function**	**OS**
Schneider Electric	PLC	Leak Control	M1EV

In this example, each row represents a sentence, and each column represents a word in that sentence. For consistency in naming conventions, we used our library function to make each column value lower case, remove special characters, and insert underscores where spaces exist, which produces the following sentence "shneider_electric plc leak_control m1ev" derived from the example in table 1

With our rows being treated as NLP sentences, we were able to use word embedding algorithms to create vectors of our labels, with the context being the surrounding words in our sentence. Two main model architectures are typically used for word embeddings - continuous bag-of-words[18] (CBOW) and Skip-Gram[18] (SG), but we used SG based on its robustness against resolving missing values. With SG, for example, context words can be predicted based off a single word, which is ideal in our case of imputing blank values and using context to predict its value. For our context window size, we selected a value of 3, meaning the preceding and succeeding 3 column values would provide the context for our column labels.

The word embedding algorithm used to vectorize our data was the fasttext module available via Gensim's open source library, but originated from research at Facebook AI Research[19]. The reason for our selection of fasttext other than its inclusion of SG, was due to its ability to represent words not used in its training corpus, something that other word embedding algorithms such as word2vec[18] are unable to do. Because one of our quality assessments includes identifying labels that are not represented in our training set either due to misspelling or error, fasttext was the appropriate option for our requirements. This is due to fasttext's usage of character n-grams for training word representations, which means n-character sub-words are incorporated in the embedding, which consequently allows words that do not appear in the training set to receive its own embedding representation by summing up sub-word vectors that make up the out of vocabulary word.

Figure 6 shows the process of using our library, which begins with preprocessing the values in our corpus for consistency before training and saving our fasttext model. In addition to creating our fasttext model, our library also generates a map of values-to-columns derived from the corpus in order to identify values that fall outside of their respective columns in our training set. With the creation of the fasttext model and map, a structured data set, such as our BACnet data set, can utilize our library functions to evaluate our data set and review any issues that can be corrected after querying the trained model and map, such as performing imputation, substituting unknown data labels with values used in the training corpus, and identifying mislabeled values.

2.3.2 | Experiments and Shortcomings

Our BACnet data set contained blank values, mislabeled data, and out of vocabulary (OOV) words (i.e. words not present in the training corpus), which we cleaned or corrected using our library functions. Figure 7 shows a subset of our BACnet data set which contains the previously described issues.

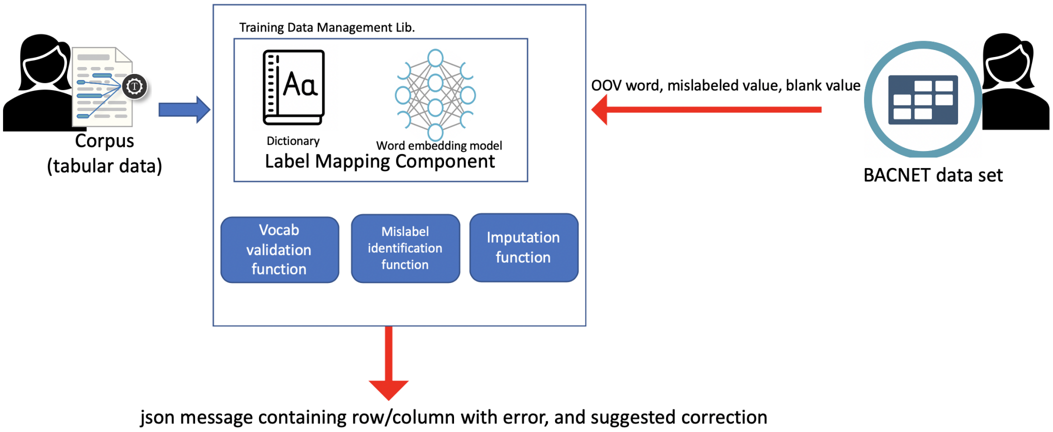
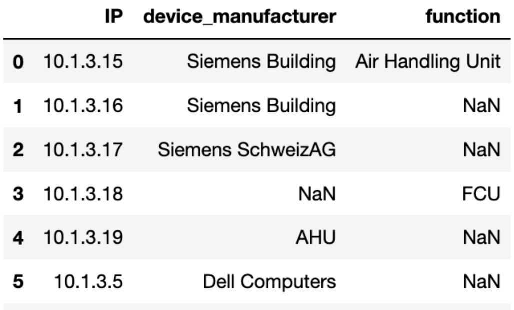

Corpus
(tabular data)
Training Data Management Lib.
Dictionary
Word embedding model
Label Mapping Component
Vocab validation function
Mislabel identification function
Imputation function
OOV word, mislabeled value, blank value
BACNET data set
json message containing row/column with error, and suggested correction

FIGURE 6 Diagram of training data management library setup



	IP	device_manufacturer	function
0	10.1.3.15	Siemens Building	Air Handling Unit
1	10.1.3.16	Siemens Building	NaN
2	10.1.3.17	Siemens SchweizAG	NaN
3	10.1.3.18	NaN	FCU
4	10.1.3.19	AHU	NaN
5	10.1.3.5	Dell Computers	NaN

FIGURE 7 Subset of BACnet data set, listing the device manufacturer and functionality

Looking at row 2 for example, our library can detect that Siemens SchweizAG is not a manufacturer found in the training corpus, so the most similar in vocabulary word is returned - in this case, the most similar vector is Siemens Building. From this and other recommendations for OOV values, we applied the changes and checked for any potential mislabeling. Row 4 in this instance reports an error for the device manufacturer column, as AHU is identified as being a function. Based on this report, we moved AHU to the correct column on this row. Finally, our data set contained many instances of null values, seen as NAN in our data set and reset to "n_a" for processing purposes. Running our imputation function on our data set offers statistically likely substitutions based on the context surrounding the null column, as well as the co-occurrence percentage of the suggested context. The blank value in device manufacturer row 3, for example suggests Schneider Electric as the manufacturer, as it appears 100% of the time as a manufacturer for fan coil units (FCU).

While our experiments returned good results for the most part, there were some shortcomings in the results, particularly with the values returned from our model's similarity function. For example, although Siemens SchweizAG on row 2 was found to be most similar to Siemens Building as we expected, its similarity score was found to be only 47%, which isn't the strongest result. Additionally, resolving Air Handling Unit and AHU to the same label proved difficult due to the n-grams being too distant. We believe that most of our results would be improved with a larger corpus and would yield stronger similarity scores.

2.4 | Feature Engineering

In the next sections, we explore how new features are engineered using a weighed categorical encoding technique followed by a PCA analysis to learn important features in the data set. In both cases, we show examples to highlight the effectiveness of our method.

BACnet/IP and the underlying UDP protocol provide a number of attributes that are encoded as nominal values. While some of the attributes encoding allow us to directly extract information that is associated with the observed device, our goal is to demonstrate how additional features can be engineered and used to train an ML model in order to infer information that is otherwise unavailable to the passive observer of the network. For example, the NPDU command I-Am-Router-To-Network clearly states that the device is a BACnet router to another BACnet network. Similarly, the payload from the I-Am APDU message can be used to extract the identity of the device vendor. While extracting such information from the PCAP can be useful, our interest lies in our ability to engineer new features that can be fed into an ML model and allow us to infer information about the device that is not explicitly stated in the BACnet stream.

While an in-depth knowledge of BACnet protocol is not required, some basic understanding of its features are needed in order to formulate some initial hypothesis to frame our feature engineering work:

1. BACnet and all UDP protocols have one or more assigned ports. It is uncommon to find the same port being used by more than one protocol unless it is dynamically assigned by the application. Therefore we should be able to use *UDP Port* as a way to isolate BACnet traffic from the rest of the network traffic in any data set.

2. BACnet-APDU *Confirmed Service* reflects the function and role of the device. When used in an ML model, it allows us to build advanced analytics that use the *Confirmed Service* code to build a classification model for identifying the BACnet controllers that are in the network.

3. *Object Types* are commonly use when a device provides a description of its capabilities and various readings of the objects it manages. *Object types* can be used to derive new ML features that differentiate between different types of controllers used in different applications such as power management vs. building access for example.

Using categorical encoding, each of the following attributes, *UDP Port*, *Confirmed Service* and *Object Type* is converted into its own respective feature set using the associated nominal value as a category similar to One-Hot encoding techniques. However, unlike One-Hot encoding where the values of each cell is a 0 or 1, in our case, the values assigned for each feature is weighed by the amount of traffic associated with each category. This could be the number of packets sent or received on a given *UDP Port* or the number of messages originating from the same device and referencing the same *Confirmed Service* or *Object Type*. This weighing technique is essential in our ability to identify different classes of device that operate on the same port or use similar services or manage objects of similar types.

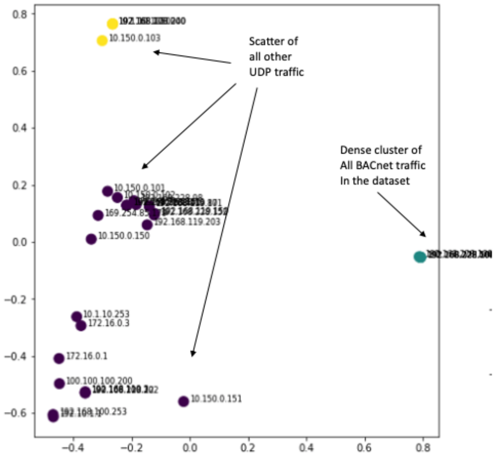


FIGURE 8 Using UDP Port number categorical encoding to identify clusters of IP devices with similar features, including a BACnet devices cluster

While *Confirmed Service* has a finite number of numerical values where each value can be converted into its own category, *UDP Port* and *Object Type* present a particular challenge. *UDP Port* is a 16-bit integer that, theoretically, can have a value between 0 and 65535. In practice, a typical one day PCAP file would show anywhere between 2000 and 3000 different port numbers used by various UDP traffic including BACnet/IP, DNS, DHCP, LLMNR, etc. A significant number of port numbers are dynamically assigned by the application or underlying IP stack running on each of the connected devices. While some port numbers provide meaningful information for differentiating different type of traffic, others simply add noise and negatively impact the performance of any ML model. In order to reduce the noise, we use a statistical approach to determine what ports are most frequently used across all the devices in the PCAP and only consider a subset of those port numbers that are used by the majority of devices. The clustering technique in Figure 8 considers only the port numbers that are used by 95 percent of the devices in the PCAP file. This leads to a 20 folds reduction in the number of unique ports, and therefore the categories, used for our training.

Figure 8 highlights the effectiveness of our approach when using simple k-mean clustering techniques based on engineered features for *UDP Port*. Each of the devices in our data set is represented by a single vector representing the port numbers identified as categorical features and weighed by the number of packets that each device had sent or received on that port. For all intents and purposes, we set the number of clusters to 3 and use PCA-2 projection technique to project the clusters on a 2 dimensional graph. This approach does not make any assumptions on the specific port that each device is using: it simply attempts to find associations between any number of devices and any number of ports. Using this technique, we can clearly identify in the same cluster all of the BACnet devices that reference similar ports, while other UDP traffic such as DNS, DHCP and LLMNR falls in two different and distinct clusters.

2.5 | Feature Learning

Heatmaps, PCA, and k-mean clustering are primary tools in our toolkit for learning relevant features of the BACnet protocol. We use heatmaps to visualize the patterns in the data as they relate to the *UDP Ports* used by the BACnet devices, the *Confirmed Services* they reference and the *Object Types* associated with them. Given the high dimensionality of the data where the number of features can easily exceed the capabilities of most visualization tools, PCA techniques allow us to narrow down the list to identify the smallest number of distinguishing features. Using the smaller number of identified features, clustering techniques are then used to confirm the feature set for training.

While the BACnet standard defines a finite number of *Object Type*, in practice, many device manufacturers introduce their own encoding for new *Object Types* that are specific to their devices. Much like *UDP Ports*, those added *Object Type* values are seldom used by more than one or two devices in the network and can be dropped from the list of categories for *Object Types*. While this technique provides some reduction in the number of categories, it significantly reduces the amount of noise in the training data.

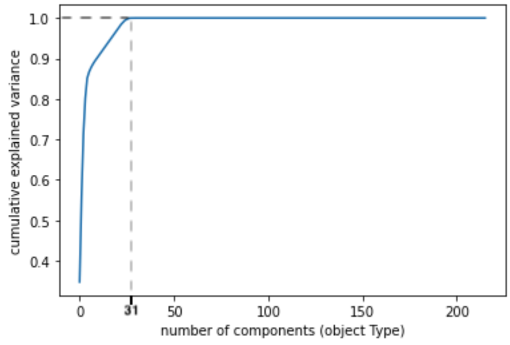


FIGURE 9 PCA Analysis of BACnet Object Types reveals: 31 components (out of a total of 216) describe 100% of the training data

Figure 9 shows the results of applying PCA analysis on BACnet *Object Type* features. A typical PCAP capture of BACnet/IP traffic can expose over 200 different *Object Types* used by the various devices connected to the network. Using PCA analysis, we can reduce that number down to 31 learned features. This reduction in number of features significantly reduces the noise in the training data, reduces the time required for training the ML model, and significantly improves the precision of our algorithms.

This approach shows equal benefits when using shallow models as well as deep learning algorithms such as a Binary Neural Network (BNN).

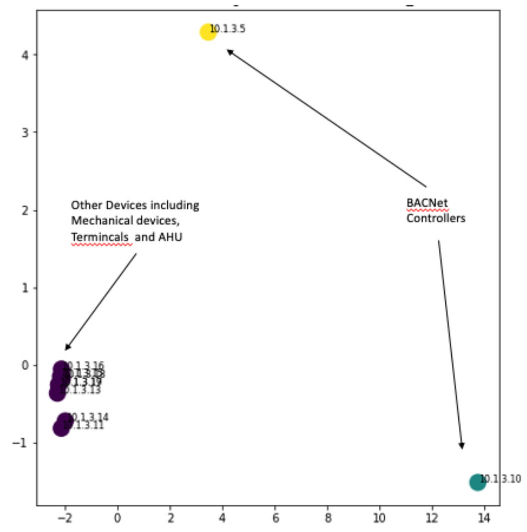


FIGURE 10 BACnet device clusters based on Confirmed Service identifies devices with similar functions

Once the appropriate number of features is selected, we run a K-mean clustering algorithm to confirm a proper separation of BACnet devices based on learned features.

Figures 10 and 11 show different clustering of the training data driven primarily by feature selection: In one instance, the selected features for *Confirmed Service* allow us to isolate devices with different roles such as Windows workstations, while in the second instance, the selected features for *Object Type* allows us to group BACnet devices that have similar objects such as two PLC from the same vendor.

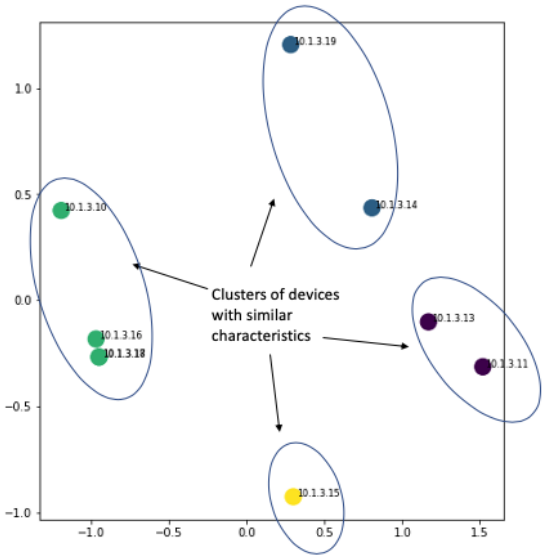


FIGURE 11 BACnet device clusters based on Object Types identifies devices with similar design/manufacturer

2.6 | Model Zoo

Using OCC that are trained using in-class features for each device in the training set addresses the issue we commonly find in PCAP files with unbalanced training data. At the same time, this approach causes an explosion in the number of running ML models, each trained to provide a score for each device in the PCAP file. A typical medium size building may have around 600 endpoints. This translates into 600 trained ML models and 600 scores for each of the devices in the network. In order to reduce the complexity of such systems, we introduce an ensemble method that aggregates all the scores for each device into a single score.

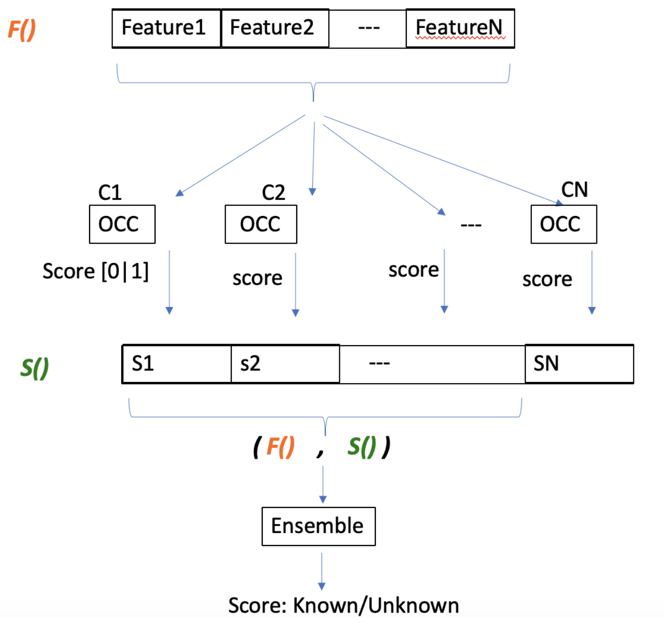


FIGURE 12 Two-stage boosting method for Model Zoo

We implement a two stage Model Zoo classifier where the first stage consists of the OCC for each of the endpoints in the training data and the second stage consists of an ensemble method that uses the scores from the first stage in order to provide a single score for each of the endpoints in the test data set. The number of epochs for training and duration of each epoch is set in relation to the volume of the traffic observed: the higher the volume of traffic, the shorter the epoch as more data is available for training in a shorter period of time.

Figure 12 shows the structure of the Model Zoo where each feature vector f() for each endpoint in the test data is presented to each of the OCC C1 to CN and the corresponding scores S1 to SN are combined together, along with the original feature vector, into a new feature vector that is presented to the ensemble classifier. As such, the ensemble classifier provides a single score for each endpoint.

When labeled training data is available, a Random Forest algorithm is commonly used for the second stage ensemble classifier. However, in many instances, a binary classifier such as Isolation Forest can also be used to provide a known/unknown score. Similarly, in most cases a shallow model for the OCC, such as Isolation Forest, OneClassSVM or an Elliptical Envelope provides good accuracy, but can also be extended to use deep learning techniques such as BNN that provides greater resiliency against missing features that are present in the training set but missing in the test set, as is often the case when training a model based on data from one building and testing it on data from a different building.

3 | RESULTS

We obtained results from experiments we conducted using OCCs both as a standalone and with a Model Zoo architecture, as well as initial results via deep learning techniques. The goal was to identify specific endpoints on the network through their behavior, which we refer to as *knowns*, while also being able to distinguish between other known endpoints and endpoints whose behavior was unclassified, referred to as *unknowns*. Reviewing the results from these techniques shows how the feature set used to define device behavior affects the performance of OCCs, but that boosting techniques such as Model Zoo can improve the overall scores regardless of the trained feature set.

3.1 | Statistical Machine Learning: Binary and Ensemble Classification

In order to identify certain endpoints on the network, we utilized an array of binary classifiers to learn the behavior of specific devices. Binary classifiers are unsupervised learning models, meaning there's no labels used during training, and are capable of classifying only a single class. Therefore, any sample that a binary classifier predicts as one that it was not trained on is deemed an outlier, or unknown.

For our experiments, we used a data set of SCADA network activity covering the period of six days, with each day aggregated over hourly epochs of invoked functions. For our training set, we used 80% of the data, with the remaining 20% used for testing. Our data set consisted of 7 known endpoints: 10.1.3.10, 10.1.3.11, 10.1.3.13, 10.1.3.17, 10.1.3.18, 10.1.3.19 and 10.1.3.5

In order to identify these specific endpoints, we trained two different types of binary classifiers, Isolation Forest, and Elliptic Envelope. We also experimented with two different encoded BACnet functions for our feature vectors – *Confirmed Service* and *Object Type*. The goal was to determine if the binary classifier arrays were sufficient in identifying the behavior of the specific endpoints they were trained to learn, and identify all others as unknown.

After training, we tested our binary classifiers on the hourly epochs contained in our test data set, with some of the results shown in Figures 13, and 14.

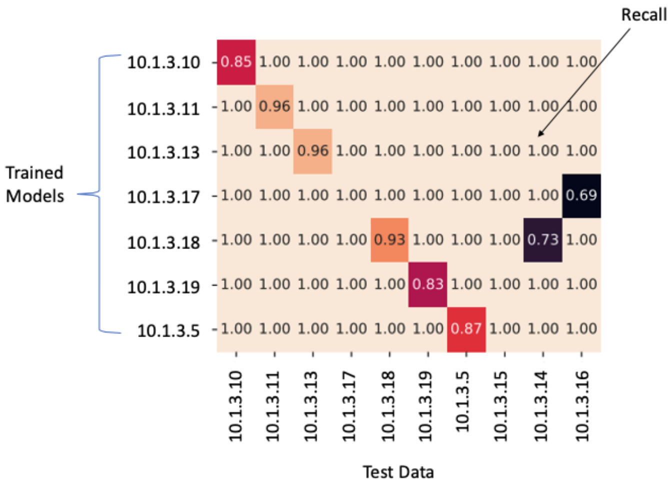
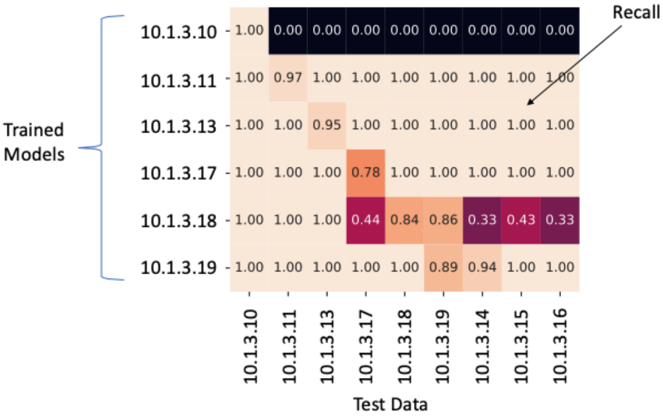


FIGURE 13 Confusion matrix for known/unknown classification results with Elliptic Envelope model trained on Confirmed Service features. Ideal matrix for binary classification is 1.00 across all columns, demonstrating perfect recall for known and unknown endpoints

Figure 13 shows the confusion matrix for predictions of the Elliptic Envelope model when trained on the *Confirmed Service* function features. The y-axis represents the predictions of the known endpoint classifiers, and the x-axis represents all encountered endpoints in the network that was scored by each known classifier. Examining the x-axis, we can see 3 unknown endpoints, 10.1.3.14, 10.1.3.15, and 10.1.3.16, were contained in the data set. Ideally, we would like to see 1.00 across the matrix, as this would mean 100% recall from the classifiers, which is the proportion of actual positives identified correctly. From the matrices, we can see that most classifiers were successful in identifying true negatives (i.e. unknowns). However, we can see there were



	10.1.3.10	10.1.3.11	10.1.3.13	10.1.3.17	10.1.3.18	10.1.3.19	10.1.3.14	10.1.3.15	10.1.3.16
10.1.3.10	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10.1.3.11	1.00	0.97	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10.1.3.13	1.00	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
10.1.3.17	1.00	1.00	1.00	0.78	1.00	1.00	1.00	1.00	1.00
10.1.3.18	1.00	1.00	1.00	0.44	0.84	0.86	0.33	0.43	0.33
10.1.3.19	1.00	1.00	1.00	1.00	1.00	0.89	0.94	1.00	1.00

FIGURE 14 Confusion matrix for known/unknown classification results with Isolation Forest model (15 estimators) trained on Object Type features. Ideal matrix for binary classification is 1.00 across all columns, demonstrating perfect recall for known and unknown endpoints

also some false negatives when identifying its own class. Although the Elliptic Envelope scored better than the Isolation Forest classifier, both models had errors with the binary classifiers for .17 and .18, with these classifiers producing false positives for one or more unknown endpoints in the data set.

On the other hand, Figure 14 shows the confusion matrices of the Isolation Forest model when trained on the *Object Type* function features. Compared to the matrices produced from the *Confirmed Service* features, we can see that two classifiers in particular, .10 and .18, produced a high number of false positive identifications on all other devices, which suggests there's not enough distinctive behavior from these two endpoints when invoking this particular function.

In order to boost the binary classifiers' capabilities, we added a Model Zoo to see if we could more accurately classify the known endpoints. To test our Model Zoo, we split the data set into 3 partitions - 60% for training the binary classifiers, 20% for building the Model Zoo ensemble data set, and the remaining 20% for validating the Model Zoo. For the Model Zoo ensemble, we used a Random Forest classifier with 100 estimators, and a max-depth of 10. After training our ensemble with the correctness matrix built from the binary classifiers' scores on the test data set, as well as the original feature vectors from the test data set, we scored the Model Zoo on the validation data set, with some of the results shown in Figures 15 and 16.

Unlike the confusion matrices for the binary classifiers where the ideal matrix would be 1.00 all across, for the Model Zoo ensemble, which is a multi-class classifier, the ideal confusion matrix would be 1.00 only where the boosted classifier on the y-axis matches the encountered endpoint on the x-axis, and 0.00 everywhere else, as this would mean perfect recall on the ensemble's ability to predict the encountered endpoint, and that no false positives were produced. When examining our figures for the Model Zoo confusion matrices, we can see that this is the case for almost all Model Zoos, regardless of the underlying binary classifiers or input features. The only instance where there's a noticeable portion of false positives was the matrix for the Model Zoo with the Elliptic Envelope binary classifier, where .10 mistakenly identified as .18 20% of the time. However, in spite of this, it clear's from the results that the Model Zoo is advantageous for significantly improving the classification of known endpoints, as well as distinguishing known endpoints from unknown endpoints.

3.2 | Deep Learning: Initial Results

In addition to the utilization of shallow learning models used for classification of network behavior as described previously, deep learning techniques outside of the Model Zoo were also used within our work in order to classify device behavior within the network. For our experiments, a BNN was implemented with one hidden layer of 84 units with a Rectified Linear Unit (ReLu) and a second hidden layer with 2 units along with a Sigmoid activation layer due to the binary nature of the classifier. The training set for the BNN contained both known and unknowns and was balanced using random sampling. Each classifier computed a threshold probability parameter using a hold out prediction set. In order to gain a better insight into classification capabilities, both validation sets were initially sorted into knowns and unknowns. As a result, the precision and recall scores for the known and unknown validation subsets are uncorrelated. Figure 17 shows an example score report for an instance of the BNN trained using the BACnet *Confirmed Service* feature set.

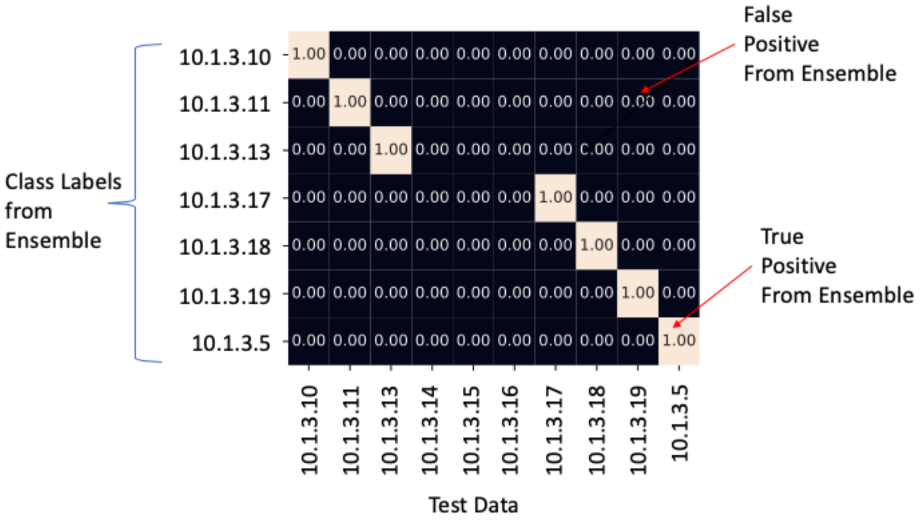


FIGURE 15 Confusion matrix for Model Zoo results with Elliptic Envelope model trained on Confirmed Service features. Ideal matrix for multi-class ensemble classification is 1.00 where matching class on y and x-axis meet, and 0.00 on all other columns, demonstrating perfect true positive and true negative predictions.

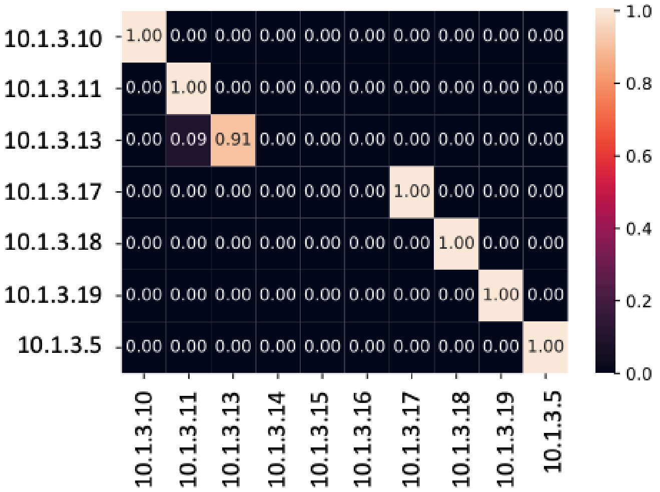
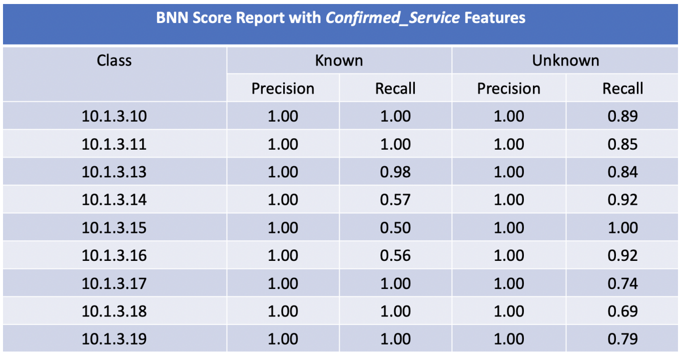


FIGURE 16 Confusion matrix for Model Zoo results with Isolation Forest model trained on Object Type features. Ideal matrix for multi-class ensemble classification is 1.00 where matching class on y and x-axis meet, and 0.00 on all other columns, demonstrating perfect true positive and true negative predictions.

Our preliminary results demonstrate that the BNN approach is in fact successful in identifying both known and unknown endpoints, as seen in the precision scores shown in Figure 17. We do however note that generally, for both knowns and unknowns, the recall scores for this classifier were worse than those for the Model Zoo built with shallow classifiers only. Finally, we note that recall scores for some known endpoints (10.1.3.14, 10.1.3.15, 10.1.3.16) demonstrate that the BNN approach using this particular feature set needs further investigation, potential tuning, and a possible increase in available training data in order to improve future results.



BNN Score Report with *Confirmed_Service* Features

Class	Known		Unknown	
	Precision	Recall	Precision	Recall
10.1.3.10	1.00	1.00	1.00	0.89
10.1.3.11	1.00	1.00	1.00	0.85
10.1.3.13	1.00	0.98	1.00	0.84
10.1.3.14	1.00	0.57	1.00	0.92
10.1.3.15	1.00	0.50	1.00	1.00
10.1.3.16	1.00	0.56	1.00	0.92
10.1.3.17	1.00	1.00	1.00	0.74
10.1.3.18	1.00	1.00	1.00	0.69
10.1.3.19	1.00	1.00	1.00	0.79

FIGURE 17 Score report for instance of BNN using only BACnet Confirmed_Service feature set

4 | DISCUSSION & CONCLUSION

To summarize, the key contributions of this work are as follows:

- demonstration of NLP techniques to correct training data label errors and inaccuracies commonly associated with BMS data sets;
- implementation of a binary classifier/ensemble framework which can be used with various scoring epochs to recognize both endpoint behavior (by IP address) and known and unknown classification in a BMS network; and
- introduction of a viable probabilistic deep learning approach with a Bayesian Neural Network for accurate classification of known and unknown endpoints in a BMS network.

In the future, we hope to further explore and employ more deep learning methods for classification. More specifically, we hope to better understand the properties of BNNs [20] for later implementation into our Model Zoo framework. We believe that implementation of this network as a probabilistic classifier will further improve our model's ability to handle unknowns. While initial results for our current implementation of this network using learned features from the BACnet protocol have indeed been promising, more comprehensive studies are needed in order to better determine the payoff between the network's computational complexity and its classification accuracy. Furthermore, in the future, much more thorough investigation must be done regarding the features used to train networks and classify results in order to avoid overfitting as a result of noise present in the features. Finally, given the complexity and breadth of the BACnet protocol, we hope to employ autoencoders within our model for feature learning as another mechanism to reduce noise in the networks.

Additionally, along with improving our algorithmic understanding of these deep learning techniques, we hope to enhance the current Model Zoo framework in order to allow for easier implementation of a variety of neural networks into the machine learning and classification pipeline in the future. For instance, increased complexity in the training of neural networks as opposed to that of shallow methods results in the need for the API to allow hyperparameter adjustments as well as inspection of and retraining in specific parts of the network. Further data analysis should also be considered in the future, as training of the neural networks will require larger data sets which include instances of both the known and unknown classes.

Finally, work in cleansing our categorical labels can be improved by further expansion of examples in our training corpus as well as applying different similarity functions to observe the results of similar in and out of vocabulary words for substitutions in our data set, and viewing its overall effect on our model.

References

1. Bender J, Cornell University oMN. BACnet/IP. http://www.bacnet.org/Tutorial/BACnetIP/index.html; .

2. Gouda MG, Liu XA. Firewall design: consistency, completeness, and compactness. In: 24th International Conference on Distributed Computing Systems, 2004. Proceedings. ; 2004: 320-327.

3. Bandara AK, Kakas AC, Lupu EC, Russo A. Using argumentation logic for firewall configuration management. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management. ; 2009: 180-187.
4. Lin CY, Nadjm-Tehrani S. Understanding IEC-60870-5-104 traffic patterns in SCADA networks. In: Proceedings of the 4th ACM Workshop on Cyber-Physical System Security. ; 2018: 51–60.
5. Sayegh N, Elhajj IH, Kayssi A, Chehab A. SCADA Intrusion Detection System based on temporal behavior of frequent patterns. In: MELECON 2014-2014 17th IEEE Mediterranean Electrotechnical Conference. IEEE. ; 2014: 432–438.
6. Maglaras LA, Jiang J. Intrusion detection in SCADA systems using machine learning techniques. In: 2014 Science and Information Conference. IEEE. ; 2014: 626–631.
7. Ortiz J, Crawford C, Le F. DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices. In: IoTDI '19. Association for Computing Machinery; 2019; New York, NY, USA: 106–117
8. Yang H, Cheng L, Chuah MC. Deep-learning-based network intrusion detection for SCADA systems. In: 2019 IEEE Conference on Communications and Network Security (CNS). IEEE. ; 2019: 1–7.
9. Kremer J, Sha F, Igel C. Robust active label correction. In: International Conference on Artificial Intelligence and Statistics. ; 2018: 308–316.
10. Nicholson B, Zhang J, Sheng VS, Wang Z. Label noise correction methods. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE. ; 2015: 1–9.
11. Fierro G, Guduguntla S, Culler DE. Dataset: An open dataset and collection tool for bms point labels. In: Proceedings of the 2nd Workshop on Data Acquisition To Analysis. ; 2019: 40–42.
12. Dietterich TG. Ensemble methods in machine learning. In: International workshop on multiple classifier systems. Springer. ; 2000: 1–15.
13. Alerton Technologies B. S. oI. The Language of BACnet-Objects, Properties and Services. http://www.bacnet.org/Bibliography/ES-7-96/ES-7-96.htm; .
14. Paparrizos J, Gravano L. k-shape: Efficient and accurate clustering of time series. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ; 2015: 1855–1870.
15. Thalheim J, Rodrigues A, Akkus IE, et al. Sieve: Actionable Insights from Monitored Metrics in Distributed Systems. In: Middleware '17. Association for Computing Machinery; 2017; New York, NY, USA: 14–27
16. Bhattacharjya D, Subramanian D, Gao T. Proximal graphical event models. In: Advances in Neural Information Processing Systems. ; 2018: 8136–8145.
17. Project Haystack. https://www.project-haystack.org; .
18. Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. https://arxiv.org/pdf/1301.3781.pdf; 2013.
19. Bojanowski P, Grave E, Joulin A, Mikolov T. Fair Open Sources fastText. https://research.fb.com/blog/2016/08/fasttext; 2016 (accessed July 28, 2020).
20. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 1998; 86(11): 2278–2324.