SAND2017-1752C

Exploiting Time and Subject Locality for Fast, Efficient, and Understandable Alert Triage

David Kavaler, Corey Hudson, Michael Bierma
Sandia National Labs[*]
Livermore, California 94551-0969
{*dkavale, cmhudso, mbierma*}@*sandia.gov*

Abstract

In many organizations, intrusion detection and other related systems are tuned to generate security alerts, which are then manually inspected by cyber-security analysts. These analysts often devote a large portion of time to inspecting these alerts, most of which are innocuous. Thus, it would be greatly beneficial to reduce the number of innocuous alerts, allowing analysts to utilize their time and skills for other aspects of cyber defense. In this work, we devise several simple, fast, and easily understood models to cut back this manual inspection workload, while maintaining high true positive and true negative rates. These methods can be introduced with nearly no overhead, and can be fine-tuned to desired true positive, true negative, and manual inspection workload requirements. In addition, due to their simplicity, these models' predictions can be easily understood and are therefore easier to trust. We demonstrate their effectiveness on real data, and discuss their potential utility in application by others.

1 Introduction

Many organizations, especially those that handle sensitive information, have a cyber-security team that is responsible for monitoring and assessing threats on their network. These *security analysts* are highly skilled experts, trained specifically for this job. However, there are only a finite number of analysts who can perform this task adequately, and often their workload is vast. These individuals must, among other tasks, remain on the forefront of new known attacks, develop new methods for detecting yet-unseen threats, and attend to current potential threats. These teams often face an overwhelming volume of alerts, most of which are innocuous, but must still be addressed. Due to this enormous volume, analysts may experience "alert desensitization" [1]; due to the number of false positives encountered, analysts may miss true positive alerts, which lead to, *e.g.*, compromised internal systems. In fact, alert desensitization has been used to describe the failure of Target to recognize an attack that led to the leak of millions of credit card numbers [2]. Thus, analysts would greatly benefit from a system that can filter (or *triage*) innocuous alerts and bring attention to those which may be real attacks. In addition, organizations would likely prefer their skilled security experts to use their knowledge to its fullest potential, performing the most difficult of tasks (*e.g.*, mentioned above, developing methods for detecting unknown attacks), rather than spending valuable time and resources attending to an alert queue.

Prior research has made headway into performing this alert triage task [3]. However, their methods require building machine learning models based on prior detected threats, which may take a long time to construct and use. Security threats rapidly evolve and change, which necessitates continuous updating of these models. This can be done, but often with a heavy cost of time and computing resources. In addition, due to the way these models are built, the task of classifying a previously unseen attack is difficult.

Machine learning models are also notoriously difficult to infer from *i.e.*, how or why does this model work, and can its predictions be trusted [4]? Such questions are vitally important in cyber-security, as the ability to understand and trust a model's predictions can be the difference between a critical failure and a large reduction in mundane workload and increase in quality-of-life for your analyst team.

There is a general dearth of research on the alert triage task in the security community, likely due to the nature of most alert triage data; it is most often proprietary, and understandably organizations are reluctant to openly re-

[*]Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

lease such data to the research community.

In this work, we present a number of simple, fast, and easily understood models to classify alerts and perform alert triage with high accuracy. These methods require no explicit prior knowledge of attacks on the system, and exploit attack time- and subject-locality. They can be tuned to an organization's specific needs depending on the acceptable amount of false positives and negatives, while reducing the number of alerts that require manual inspection. We test these models on real cyber-security data from our organization, and find:

1. The simplest model, which reduces the number of alerts an analyst must manually inspect by 50%, yields 97.18% true positive and 99.84% true negative rates. Minor augmentations can increase the true positive and true negative rates to 98.38% and 99.90%, respectively, reducing manual inspection workload by 48.40%.

2. More advanced (but still simple) models provide up to 99.15% true positive and 99.96% true negative rates, with a 99.32% workload reduction. Depending on how much one wishes to reduce manual inspection workload, true positive and true negative rates can be increased. We provide a few trade-off curves that allow management of true positive and true negative rates, balanced by manual inspection workload reduction, that can be tuned to an organization's specific needs.

3. Our methods work because our data contains alerts that are grouped by subject and time (*i.e.*, locality), and the groups generally have homogeneous labels (*i.e.*, alerts within a group are often labeled the same). We discuss the applicability of our work to other data that may be more heterogeneous within subject- and time-localities.

In addition, we provide our data set in an anonymized form[1] in hopes to increase transparency of our methods and in hopes that the data can be used for further research in this field.

The rest of the paper is organized as follows: Section 2 covers data and methodological goals; Section 3 discusses related work; Section 4 presents results; Section 5 provides a discussion; Section 6 covers threats to validity and future work; and Section 7 provides a conclusion.

2 Data and Methodological Goals

Figure 1 illustrates the alert generation process for our data. Alert data analyzed by the security team is generated through multiple sensors, which may have different data formats. This raw data is then transformed into a unified format, and structured based on various meta-attributes to a human-digestible form. Here, our "structuring filters" are written by real analysts to group raw data by *subject*. Raw data is collected in time-batches; the period of the time-batch depends on the subject. Subjects can be thought of as a set of rules defined by the analyst team to detect and categorize particular threats. The approach for alert generation in our data is common in network monitoring.

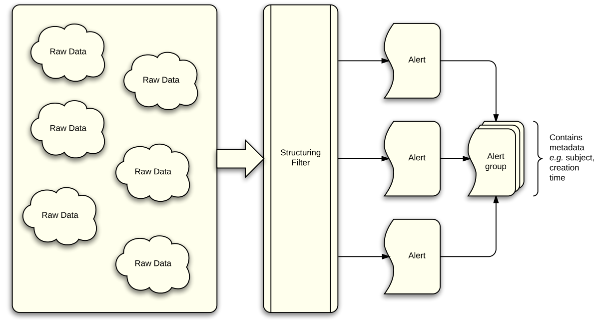


Figure 1: Alert generation process. Raw data is sent through structuring filters written by security analysts. These structuring filters turn the raw data into an easier-to-read form for later labeling. Alerts generated by the same filter within the same time-batched run are combined into alert groups.

For example, one of our filters looks for all HTTP requests from a particular IP block, and runs twice a day[2]. If there are 5 instances of this event in the first run, and 3 instances in the second, we will generate two *alert groups*: the first comprising of 5 alerts (corresponding to instances), and the second comprising of 3 alerts. In this way, we have *subject*- and *time-locality*; each alert group contains only alerts of a given subject (in this example, HTTP requests from a particular IP block), which are grouped in time. These alert groups also contain meta-data; this meta-data is not analyzed in this work.

After these alerts have been generated and grouped, analysts manually inspect each alert individually and mark them as "closed" (*i.e.*, innocuous) or "promoted" (*i.e.*, a potential threat that requires additional attention)[3]. Our goal is to build a classifier which accurately labels alerts as "closed" or "promoted" without requiring extensive manual inspection. Note that we assume all analysts label alerts correctly; in other words, our ground

[1]The anonymization does not affect the results presented in this work.

[2]The subject of this alert type would then be of the form: "HTTP requests from external IP block X"

[3]Analysts may also leave alerts "open". We remove these from our data set as there are a myriad of reasons by which an alert will remain open.

truth for all classification methods in this work are labels provided by a real, working analyst team. The assumption that analysts are always correct may not be true in practice, but is a necessary assumption as we use their labels as our ground truth. It would be very difficult to construct a classification model of any type without this assumption.

In this work, we focus on exploiting only time- and subject-locality of alerts. As mentioned above, these alert groups do contain additional meta-data. The natural question then is: why not use meta-data as well? A primary point of consideration in this work is to construct *understandable* models. We could very well extract every feature we can imagine and use those in addition to time and subject features. However, we believe this would severely hamper the ease of understanding our models, and potentially violate the principle of parsimony [5], commonly observed in statistical modeling [6][4]. It is critically important that we are able to understand and trust our models' predictions. Using a large number of features can easily make this task unwieldy. In addition, increasing the number of features also increases the time it takes to build a model. In essence, we have four main methodological goals for our classification models; they must:

1. Have a high true positive rate; secondarily, have a high true negative rate. Missing an alert that should have been promoted (*i.e.*, false negatives) can be extremely costly, and should be avoided with priority.

2. Be fast to build, and fast to apply for prediction.

3. Be simple to understand, and thus easier to trust.

4. Save analysts' time and resources (*i.e.*, reduce the number of alerts an analyst must manually inspect).

In addition, we must make sure that there are no problems of *data leakage* [7], *e.g.*, future data being used to predict data in the past. Our models are explicitly made and evaluated such that this time ordering must be respected and maintained.

We evaluate our models using true positive rate (*sensitivity*) and true negative rate (*specificity*) for a number of reasons. AUC (area under the receiver operating characteristic curve) – a standard evaluation metric for classification models [8] – does not translate well for the models presented here. In essence, AUC measures the area under the curve between true positive rate and false positive rate, for varying probability thresholds of a classifier [9]. Our models do not provide a prediction probability; they output predictions with 100% confidence. Thus, AUC

[4] *i.e.*, Occam's razor.

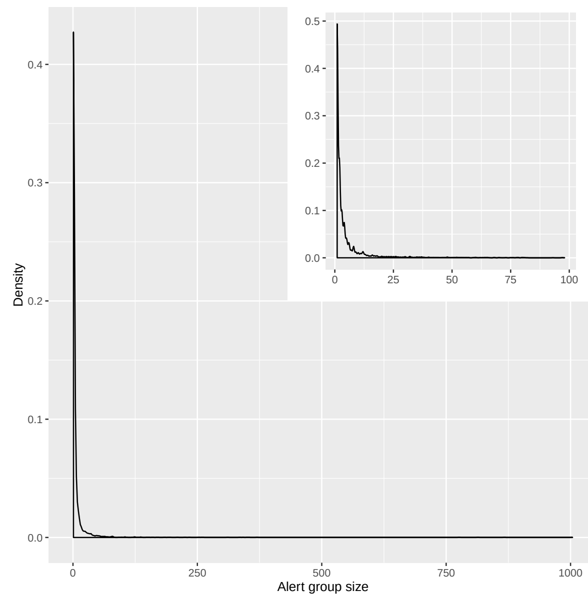


Figure 2: Distribution of alert group sizes. Inset depicts computed density for group frequencies between 0 to 100.

cannot be directly used for our models. Another potential evaluation metric is the overall accuracy (number of correct predictions divided by total number of points). The issue with overall accuracy is that our data is heavily unbalanced. In fact, if we build a poor classifier that predicts all alerts should be closed, we would have a 94.87% overall accuracy on our data; clearly, this is misleading. In contrast, for this case the sensitivity would be 0% and the specificity would be 100%. Thus, sensitivity and specificity have better discriminatory power than overall accuracy. A third possible metric would be balanced accuracy. However, this is merely the arithmetic mean of sensitivity and $1-$specificity; thus, reporting both sensitivity and specificity provides at least as much information as balanced accuracy.

3 Related Work

The application of data analysis to cyber security alert classification has been well studied. Intrusion detection systems have been augmented by a variety of techniques including Bayesian networks [10], PCA [11] and decision trees [12] in order to reduce the manual efforts of human analysts and improve threat detection. While these techniques have shown promising results [13], even for the detection of previously-unseen malware [14], these techniques are much more computationally expensive than our approach and the models may need to be rebuilt frequently in order to keep up with the changing threat landscape. Additionally, it is often difficult to interpret and trust the behavior of these models, as is the case with many machine learning methods [4]. This is especially important when quantifying the future effectiveness of the models, and understanding their areas of weakness.

Although recent work has made improvements in the interpretability and performance analysis of machine learning models [4], they only provide locally interpretable results that must still be processed by an analyst. Locally interpretable results and model debuggers [15] are useful when building models and analyzing their output, but they will not reduce the number of alerts that the analyst must process. Our simple models reduce analysts' workload, are easily understood, and do not need additional tools to help interpret their behavior.

Temporal locality of security events has been studied in order to detect network [16] and host [17] based anomalies and to differentiate spam from non-spam email [18]. Although these works show that temporal locality can be used as a feature to segment benign and malicious behavior, both the network and host based approaches require a training period and models significantly more complex than ours. The segmentation of spam from non-spam emails is more closely related to our work, as the temporal stability of IP addresses was shown to be a good indicator of spam. The use of two features (IP address and timestamp) is similar to our use of alert subject and timestamp.

4 Results

In this section, we review our exploratory analysis (Section 4.1), introduce a naive baseline for comparison (Section 4.2), describe our strategies and their results on our data (Sections 4.3 and 4.4), and present a comparison of our methods to a "standard" machine learning approach (Section 4.5).

4.1 Exploratory Analysis

In constructing any sort of classification model, exploratory analysis is an important initial step. In addition to other benefits, exploratory analysis allows us to understand our data more in-depth, discovering potential points of focus which may help improve model performance.

We analyze $579,408$ closed alerts and $31,349$ promoted alerts, for a total of $610,757$ alerts. Thus, our data is very skewed; there are many more closed alerts than promoted alerts, as is expected with this type of data. Figure 2 shows the distribution of alert group sizes. As shown, the distribution is long-tailed, with most of the density lying below an alert group size of 25.

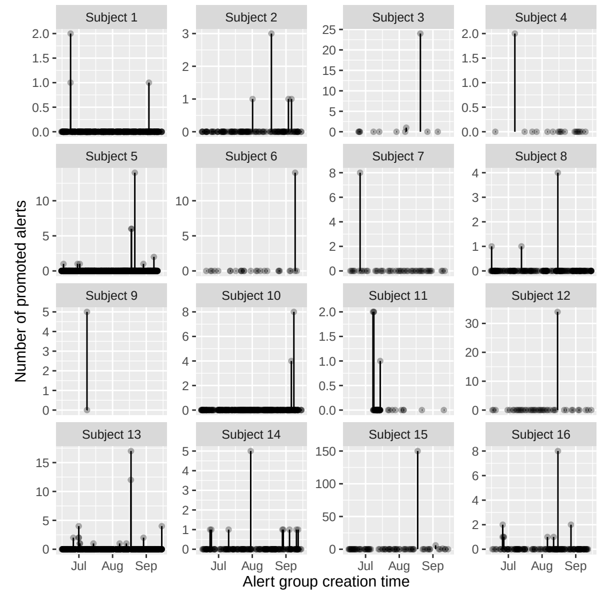


Figure 3: Number of promotions over time. Large bursts often correspond to most or all alerts within an alert group being promoted.

Initially, we set out to discover which attributes of alerts may be most predictive of their final classifications. Generally speaking, as threats evolve and change in time, we sought to discover whether or not there is clear time-based behavior in alert classifications. In addition, as skilled analysts are the ones who create the subject filters, we thought it best to examine alerts stratified by subject.

Figure 3 shows the number of promoted alerts for 16 subjects over a period of 3 consecutive months. We see that oftentimes alerts are not promoted. However, when alerts are promoted, they are commonly done so in large "bursts". For example, for Subject 3, there exists a very large burst of 23 promotions all at once. Recall that alerts are generated in groups at certain times, corresponding to alert groups. It turns out that this burst of promotions in Subject 3 is a single alert group that is all promoted[5]. We call this phenomenon one of *time-* and *subject-locality*; an alert group is built for a particular subject, at a particular point in time (corresponding to when the raw data was first encountered and processed). This discovery provides motivation. As promotions are relatively rare compared to closed alerts, if we can somehow leverage these large bursts in time, we will capture a large amount (in fact, majority) of the promoted alerts.

[5]This is a common occurrence and will be utilized in some of our models.

Table 1: Naive model confusion matrix

	Observed	
Predicted	Closed	Promoted
Closed	576907	1795
Promoted	2501	29554
	Specificity: 0.9957	Sensitivity: 0.9427

Table 2: Alternating model confusion matrix

	Observed	
Predicted	Closed	Promoted
Closed	578459	883
Promoted	949	30466
	Specificity: 0.9984	Sensitivity: 0.9718

4.2 Constructing a Naive Baseline

When developing or applying a new method to existing data, it's important to establish a naive baseline to serve as a basis of comparison, or a goal to strive for. In our data, as mentioned above, we see clear evidence of burst behavior. Thus, an extremely naive approach would be to always use the previously observed label in time as a prediction for the next label, per subject. This method is very fast and simple; it can be done trivially in near-constant time. The confusion matrix for our naive baseline model can be seen in Table 1. As shown, this naive model performs surprisingly well; we see a true negative rate (*specificity*) of 99.57% and a true positive rate (*sensitivity*) of 94.27%. However, this method has one major flaw which violates one of our aforementioned goals: it does not reduce analysts' workload. For a past label to exist, an analyst must provide one. Thus, to implement this model in practice would still require an analyst to label every point, as every future point's prediction relies on the immediate prior label. If the prior label never changes (as would be the case here without an analyst to provide a label), we have no chance of predicting points correctly. The authors wish to emphasize that this naive model is impractical to implement in practice; it is presented merely as a motivation for the ideas behind the following, practical methods, as they build on top of this naive approach. The fact that this exploratory model works so well illustrates that time- and subject-locality is important, and exploiting this fact may prove fruitful in practical, developed models.

4.3 Alternating Analyst Labels

We see that our naive model performs well in terms of our measures of accuracy. However, it does not reduce analysts' workload, as an analyst is still required to label every prior point. A small augmentation can be made

Table 3: Alternating model (augmented) confusion matrix

	Observed	
Predicted	Closed	Promoted
Closed	578856	509
Promoted	552	30840
	Specificity: 0.9990	Sensitivity: 0.9838

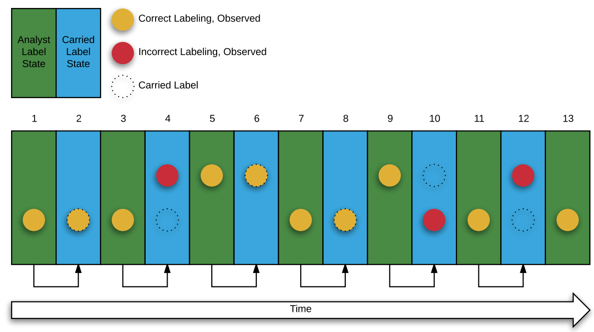


Figure 4: The process of alternating labeling. Low circles are closed labels; high circles are promoted labels. Solid circles indicate observed labels. Every other label is manually provided by an analyst and automatically carried over to the next alert in time for each subject.

which results in a sizable workload reduction: have analysts manually label every-other point in time, for each subject. Figure 4 depicts this process. Analysts manually label alerts in the *analyst label state*. These labels are then carried over for the next alert in subject-time (*carried label state*) as an automatic alert label (*i.e.*, a prediction). It can be easily seen that this reduces the number of manually labeled points by 50%, as analysts only manually inspect every-other alert[6]. The results of this process applied on our data can be seen in Table 2. These results are initially surprising; one would think that this model should be less accurate than our naive baseline, as analysts perform less manual labeling. However, this phenomenon can be simply explained.

Recall that promotions often occur in bursts (Figure 3). In Figure 4, this corresponds to a long "chain" of promoted points. In the fourth label state, we see that the first promotion in the chain is misclassified – it should be predicted as promoted, but is predicted closed. In the ninth label state, we see a *one-off* *i.e.*, a promotion that does not occur as part of a chain. The naive model would never predict this point correctly, as the prior label (eighth label state) is closed, and the naive method would carry the closed label, resulting in an incorrect

[6]This only holds if there are no subjects with only one alert. This is true for the vast majority ($> 99\%$) of our data.

prediction; however, the alternating label method classifies this alert correctly. In fact, this situation is analogous to the fourth label state; the naive model would never predict this label correctly, as the prior label is closed. In addition, the naive model would never correctly predict the label in state seven, as its prior label is different. In summary, the naive model will never properly predict one-offs, the first label in a promotion chain, or the label immediately after the end of a promotion chain. The method presented in this section has a chance to properly predict both of these labels.

To see this, consider the following. Our method labels only 50% of the points manually, using predictions to fill in the rest. If we assume that promotion chains and the number of labels between consecutive chains are independent, we have a 50% probability of landing on the first label (or after-last label) in a promotion sequence during a manual label state. This leads to an expected value of 50% of these labels being correctly classified by this model[7]. Compare this to the naive model, which never guesses these two label types correctly. Thus, interestingly, by introducing a level of uncertainty (*i.e.*, only manually labeling half of the alerts), we gain increased accuracy and reduced workload – two of our primary goals. In addition, this method is still very simple and thus easy to understand, and can also be implemented with trivial overhead.

4.3.1 A Small Augmentation

Though one-offs are a special case of the first promotion in a chain as described above, they are interesting in their own right. Why do these points exist, when chains are empirically more common? Recall that alerts are put into alert groups based on subject and time. Our data shows that one-offs in their own alert group (992) are more common than one-offs within a larger alert group (160) (*i.e.*, alert groups with both promoted and closed alerts). If we enforce by process that all alert groups of size 1 be manually inspected, we increase our true positive and true negative rates with a small loss in workload reduction (50% to 48.40%). The results of this augmentation on our data can be found in Table 3. This small augmentation to the above process meets all our aforementioned goals, and is likely worthwhile as a small loss in workload reduction is negligible compared to the gain in true positive rate.

4.4 Alert Group Fractional Sampling

Recall that alerts are grouped into *alert groups* based on subject and the time at which the raw data was scanned.

[7] One-offs can be seen as a special case of identifying the first label in a promotion sequence, and are thus covered by this analysis.

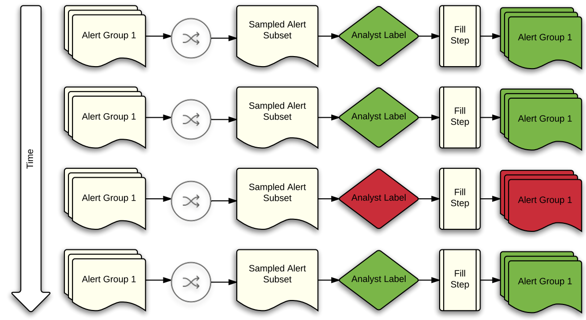


Figure 5: Process diagram for the fractional sampling model. For each alert group, a fractional subset of alerts are randomly sampled and manually labeled by an analyst. The manual labeling is then used to fill the rest of the group through one of our described strategies.

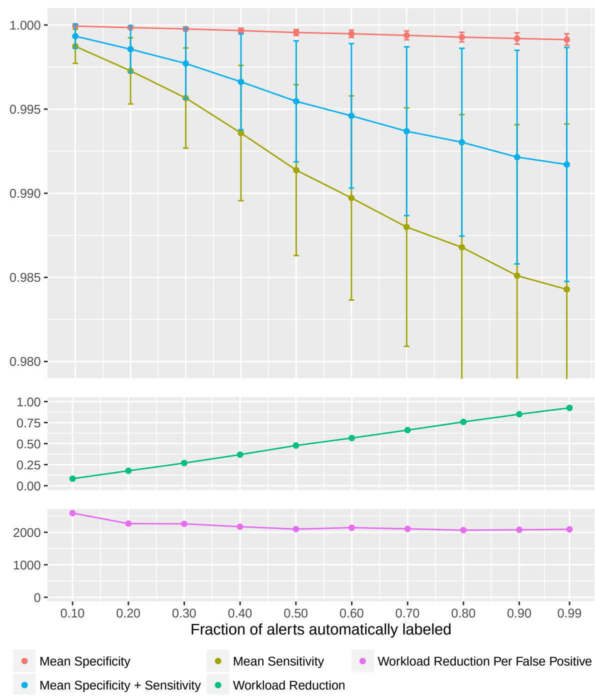
In the models presented so far, we consider time as continuous and ignore the structure imposed by alert groups themselves *i.e.*, the fact that all alerts within an alert group are considered as occurring at the exact same time. In the following models, we utilize this fact.

In all models that follow, we perform a *sampling* of a fraction of alerts within each alert group, manually labeling only the sample. Based on a particular *fill* strategy, we label the rest of the alerts in the alert group automatically, using the manually determined label as a guide. This process is depicted in Figure 5.

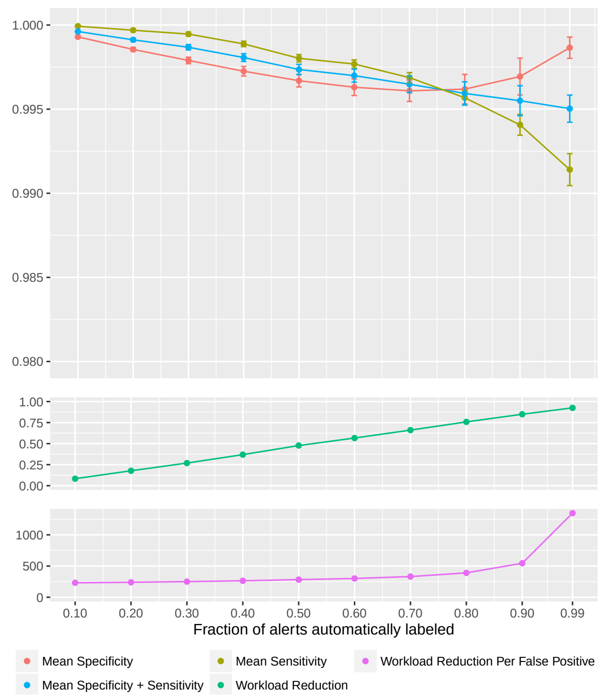
The theory behind this method is based on a few observations. As noted briefly above (Section 4.1), most alert groups contain alerts which all have the same label, *e.g.*, if one alert is promoted, it is likely all others are promoted. In other words, the conditional probability of some alert within an alert group being promoted, given another alert in the group is promoted, is very high. As we sample more alerts within an alert group, the sampling distribution of alert labels will approach the true distribution. Given our knowledge of the nature of our data, we believe this sampling distribution will approach the true distribution rapidly. This means that we can potentially sample and manually inspect a very small number of alerts from an alert group and have a good guess as to how the rest of the alerts should be labeled.

4.4.1 Fill Strategy: Random Within Sample

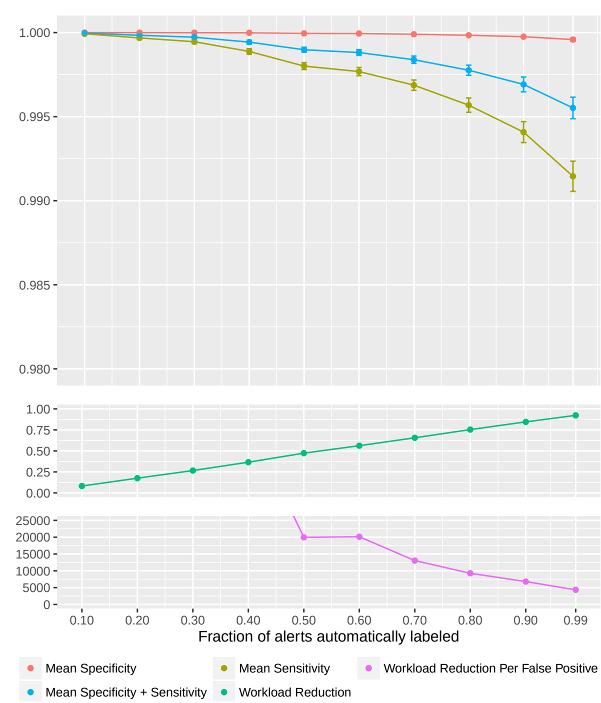
The simplest fill strategy is to take a random alert from our sampled fraction of alerts, and apply this label to all other currently unlabeled alerts within the group. For example, if an alert group has 10 alerts, and our fractional sample is set to 50%, we will manually label 5 of these alerts. We then randomly select one label from these 5 alerts and apply its label automatically to the remain-



(a) Random within sample



(b) Any promoted



(c) Heterogeneous group detection

Figure 6: Specificity, specificity + sensitivity, and sensitivity, workload reduction, and workload reduction per introduced false positive for each fraction of automatically labeled alerts, by label fill strategy. Each strategy was performed 1000 times, and plotted by mean. Vertical bars depict standard deviations. Note differing y-axes between and within subfigures. For Subfigure c), bottom row, y-values go to infinity as specificity becomes 1.

ing unlabeled alerts within the group. This second random selection step may seem superfluous, but is necessary to deal with *heterogeneous groups i.e.*, alert groups that contain both promoted and closed alerts. If an alert group is heterogeneous, then randomly selecting one label from our initial fractional sample should capture this heterogeneity on average. The results of this strategy are shown in Figure 6a.

4.4.2 Fill Strategy: Any Promoted

Another simple fill strategy is to look at our manually labeled sample and see if any alerts are promoted. If so, we automatically say the rest of the alerts in the group should be promoted. This strategy heavily biases our predictions towards increasing true positives – one of our primary goals. In addition, this somewhat follows the cognitive bias that may exist when performing labeling. If one alert within a group is promoted, it is not far-fetched to believe that extra attention should be paid to the rest of the alerts in the group. This method, however, will inevitably increase false positives. The results of this strategy are shown in Figure 6b.

4.4.3 Fill Strategy: Heterogeneous Group Detection

The final fill strategy presented is one of heterogeneous group detection. As explained above, a heterogeneous alert group is one in which some alerts are promoted and some are closed. In this strategy, if we see heterogeneous labels in our manual labeling of the fractional sample, we manually label the rest of the alerts in the group. The theory behind this method is: if most groups are *homogeneous*, this strategy works just as well as random sampling in terms of workload reduction; we do not waste time manually labeling alerts that have the same label as others in the group. However, for groups that are heterogeneous, we gain the benefits of increased true positives similarly to the "any promoted" strategy, as we manually label the entire group. The results of this strategy are shown in Figure 6c.

Figure 6 shows the results of the fractional sampling approach. The x-axis is the number of alerts that are automatically labeled (*i.e.*, $1-$fractional sample size). The first row of each plot shows specificity, mean sensitivity, and their average. Vertical bars are standard deviations over 1000 runs[8]. The second row depicts workload reduction. Note that the slope of this line for each plot is not exactly 1; there are many alert groups of size 1, and since we cannot sample a fraction of an alert, these alerts are always manually labeled regardless of fractional sample size. In addition, we take the ceiling of the fractional sample size, *e.g.*, if the alert group is size 101 and our fractional sample size is 1%, we will manually label 2 alerts. The bottom row shows workload reduction per false positive, which is meant to illustrate a trade-off between increased false positives and decreased workload. A further description of this trade-off plot follows.

[8] As this method is probabilistic, we plot the average across all runs.

Our primary goal is to maximize true positive rate. When reducing the number of manually inspected alerts, this becomes difficult and generally comes at a cost of increased false positives. False positives increase overall effort required, as alerts that are marked as promoted (whether or not they truly should be) must be reviewed, discussed, and handled beyond the initial effort of performing the labeling. Thus, if we introduce too many false positives, we may completely nullify the workload reduction we gain in terms of manual inspection time. For example, in an extreme case, say we mark everything automatically as promoted. This yields a workload reduction in initial manual inspection (as is discussed here) of 100%; however, this is practically meaningless as all these alerts now incur an additional overhead in analyst time and resources spent, as they must be handled more in-depth than if they were manually labeled as closed. Thus, we wish to strike a balance between the increase in false positives and labeling workload reduction.

The bottom row of each subplot in Figure 6, then, can be interpreted as: for each false positive we introduce due to decreasing labeling workload, we incur some additional overhead. In the case of Figure 6c at an automatic labeling level of 99%, if this overhead time is ≥ 4354 times the amount of time it takes to initially label an alert on average, then the initial labeling workload reduction is completely nullified by the increase in overhead due to false positives.

Another trade-off analysis can be seen in Figure 7. Here, we see workload reduction (x-axis) and sensitivity and specificity. The idea is that there comes some point at which we rapidly decrease sensitivity and specificity with a small decrease in workload (*i.e.*, diminishing returns). Using these plots, we can determine this point for each fill strategy. For the any promoted (green) and heterogeneous group detection (red) strategies, we generally see sudden rapid loss with respect to sensitivity at 70% workload reduction. We see something similar for specificity for heterogeneous group detection, but a rapid gain for any promoted. Interestingly, we don't see this sudden rapid loss for the random within sample method; the loss is linear across workload reduction levels, but also consistently lower than the heterogeneous group detection method.

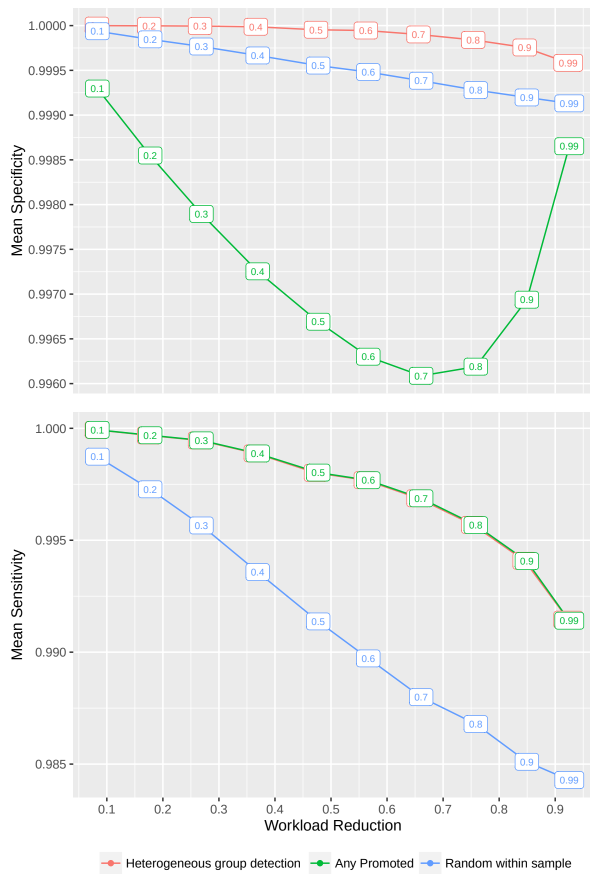


Figure 7: Trade-off between workload reduction and specificity and sensitivity, by label fill strategy. Square labels show fraction of automatically labeled alerts. Note differing y-scales.

4.5 Comparison to a "Standard" Machine Learning Approach

The classification methods outlined above can be thought of as primitive machine learning approaches. In fact, the authors' initial reasoning behind attempting these simple methods were guided by first implementing a classifier on the same data, plus meta-data, using standard machine learning methods. One such method is classification by random forests [19]. Using the Python package scikit-learn [20], we built a random forest classifier with the number of trees determined by out-of-bag error rates [21][9], evaluated using a method of time-series cross-validation sometimes called *forward chaining* [22] with 4 folds. We use only 4 folds due to the rarity of promoted alerts; using more total folds causes some folds to contain no promoted alerts. The use of *time-series* cross-validation, as opposed to a canonical cross-validation is crucially important for data with an explicit time component as discussed in Section 2[10]. To accurately evaluate the usefulness of such models, we must be careful to only train using data that occurs temporally before the test set.

Forward chaining works as follows. If we are evaluating using 4 folds, we split the data into 5 *time-ordered* parts of equal size. We then train using all prior consecutive segments and test using the next fold on the horizon. In other words, the procedure is:

Fold 1: Train [Segment 1]; Test [Segment 2]
Fold 2: Train [Segments $1,2$]; Test [Segment 3]
Fold 3: Train [Segments $1,2,3$]; Test [Segment 4]
Fold 4: Train [Segments $1,2,3,4$]; Test [Segment 5]

The evaluation metric is then averaged across folds; here, we use AUC. Over many runs of the modeling process (*i.e.*, many random forests built), this method yields and average AUC of 65.35%. In addition, *this method never identifies any true positives*; over all random forests built and evaluated, none correctly identified a promoted alert. This could be due to the rarity of such alerts. However, even when reducing the number of folds to 2, we obtain a similar average AUC across runs (65.53%). In addition, we tried bagged decision tree ensembles without the random subspace method [23], which yielded similar results[11].

It is interesting to note that the models built through machine learning had worse AUC (along with worse sensitivity and specificity), while having access to more features. In general, having more features does not necessarily lead to a better model. However, in the case of random forests, having more features increases the chance that an optimal subset of these features will be found that result in high predictive power (when using random subspaces). In addition, we also built random forests with only the features used for our developed approaches above in order to combat potential dilution of the feature space; such dilution could lead to poor performance[12]. These forests had similar performance to other random forests tried.

The poor performance of these standard machine learning approaches provides a comparison for our simple methods outlined above. To be fair, random forests and other standard approaches attempt to automatically classify every point in the test set, potentially providing comparatively more automation. In contrast, our methods require an analyst to be present throughout time. However, we believe that the drawbacks of our methods are outweighed by their benefits when compared to a standard approach as outlined here. For example, it is

[9] `http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html`

[10] Specifically the discussion on *data leakage*.

[11] The "standard" random forest implementation in scikit-learn uses bagged decision trees with the random subspace method.

[12] *i.e.* the curse of dimensionality

generally difficult to tell how even a random forest constructs its decisions. A standard method of examining feature importance in a random forest is to see how much each feature decreases the impurity in a tree, and average over all trees [19]. However, this is affected by imbalance in data set label. In addition, we believe that this interpretation of feature importance is harder to understand than the methods presented in this work; our methods work because they take advantage of natural phenomena in this type of data: time- and subject-locality.

5 Discussion

All models discussed achieve high specificity and sensitivity. This fact may be surprising given their simplicity. Below, we discuss our reasoning as to why these models work as well as they do.

The primary reason the fractional sampling models (Section 4.4) work is that alert groups are generally labeled homogeneously; only 0.0319% of our alert groups have heterogeneous labels. Thus, even when we sample only 1% of alerts in a group, we obtain high sensitivity and specificity. The reason behind the alternating label state models' (Section 4.3) performance is similar. It turns out that promotion chains and one-offs predominately correspond to singular alert groups. The root cause of these models' performance, then, is the reason behind the prevalence of label homogeneity.

The process that generates homogeneous alert group labels is difficult to grasp, and could be due to a number of factors. It could be that our data is the result of analysts who write structuring filters favoring homogeneity, *i.e.*, they write filters such that when a potential attack is detected, the associated alert group is fully promoted or closed. This does not necessarily have to be active; analysts could be doing this without realizing. Testing whether or not this is true is difficult, as we would require alert data from multiple sources and multiple analysts who are trained in different ways. Another, possibly more likely, explanation adheres to the principle of "attack surges" [11, 16, 17]; malicious actors generally perform their attacks in time-local surges. Thus, a system which prioritizes sorting data by subject-time (as ours does) would likely contain homogeneous alert groups.

Another point of discussion is the comparison of the heterogeneous group detection fill method to others in Section 4.4. Specifically, why does this method work so much better than the others? As noted in Section 4.4.3, if most groups are homogeneous, this strategy works as well as the random within sample method in terms of workload reduction; additionally, we have a comparable specificity. However, for groups that are heterogeneous, we essentially perform a slightly smarter version of the any promoted method, adopting its sensitivity and increasing specificity. Thus, the heterogeneous group detection method has an average sensitivity that is better than the any promoted method, with an average specificity that is better than the random within sample method.

We note that the heterogeneous group detection model seems to "take the cake" across all categories of accuracy and workload reduction for our data. We present the preceding models as they may be more applicable for other forms of alert data. The heterogeneous detection model works in large part due to the prevalence of homogeneity in labels, while other models may work better for other organizations' data. As the models are simple to understand, testing whether or not they should be implemented should not be a laborious task.

5.1 Adversarial Evasion

We note that all the approaches we developed in this work could be vulnerable to an adversary that injects many heterogeneous groups. As noted, our performance on heterogeneous groups is what drives our metrics of accuracy down. However, to effectively inject heterogeneous groups into the alert system, the adversary would likely need to know the filters that are in place. In addition, if the adversary already knows the exact filters in the system that generate alerts, they would be able to avoid them completely. Thus, the adversary would need to solve a much more difficult problem in order to exploit the primary weakness of our approaches.

6 Threats to Validity and Future Work

We acknowledge that our models work in large part due to homogeneity in labels within alert groups. For other sources of alert data, heterogeneity may be more common, and our models may not work as well. However, if one uses the heterogeneous group detection model and the heterogeneous groups are heavily skewed towards one label type, then the number of correctly classified instances for that label type will still be high. Thus, if heterogeneous groups are skewed towards the promoted class, then true positives will still be high, and the primary goal will be met.

Note that a "classical" machine learning approach (*e.g.* decision trees) would generally attempt to automatically classify *all* points. Our models can never classify every point automatically; we require a certain percentage of manual labels per alert group, always at least 1. Thus, we provide an alert triage solution to a problem that is "easier" than the problem a classical approach attempts to solve. However, we do not believe that this is a severe limitation of our work, as we can attain a high workload reduction (92.322%) with high specificity (99.958%) and sensitivity (99.145%) (Section 4.4.3). In addition, this approach is very similar in spirit to active learning, where a learning algorithm can query an information source for a label when, *e.g.*, it is unsure about a prediction [24]. Here, we analogously ask the analyst to perform a manual labeling for each alert group, and extend that labeling automatically.

We acknowledge that the use of human analysts labels of "closed" or "promoted" can be an issue, as their classifications may not always be correct. However, as noted prior, it would be very difficult to create an automated classifier without these labels. In addition, there's no real way of knowing whether or not their labels are correct without explicit future knowledge of their mistakes. For labels in the past, we know that misclassification is rare. In addition, these misclassified alerts are unlikely to show up in our data as misclassified, as analysts will go back and revise labels if they have knowledge the labels were wrong. The analysts are trained experts, so we trust their decisions as the ground truth.

We acknowledge the threats behind our operationalization of workload reduction. In our approach, we assume that all alerts take the same amount of time to process to get our percentage of workload reduction. We know that this is likely not to be entirely true. However, we attempt to assuage this concern by presenting trade-off plots which can be used to help determine whether or not our presented methods are useful for a given organization. In addition, if the distribution of time it takes for an analyst to label an alert is not severely skewed, the workload reduction percentage is accurate on average. Finally, even if we are overestimating the workload reduction achieved by our approaches, we still decrease analysts' workload, which we believe is significant. In the future, we do hope to fit various distributions of time taken to classify an alert and incorporate that into our workload reduction calculation.

Even without the argument of workload reduction, we believe this work has merit if merely as an observational study. We present explanations and descriptions of the underlying alert generation process. An analyst that does not know about the time- and subject- locality of alerts would still likely spend large amounts of time labeling each alert. This work as an observational study shows that one can front-load their effort for each alert group and achieve high performance.

In the future, we hope to obtain alert data from different sources to test the applicability of our methods here. In addition, we hope to perform some type of sensitivity analysis with respect to heterogeneity of alert groups – this would be accomplished if we can obtain other data sources with varying levels of heterogeneity.

We also hope to increase our specificity and, more importantly, our sensitivity. Although we can obtain a relatively high sensitivity in terms of percentage, every missed promotion incurs a potential cost; it's possible that we will misclassify "*the* alert". However, if the average cost of a missed promotion is negatively related to the amount of heterogeneity in an alert group, the promotions our models miss will likely not be too costly. In other words, if the cost of a missed promotion is less with more heterogeneity, our models' misses will be less costly.

Finally, we hope to improve our models by adding a likelihood component. Though heterogeneous groups are rare, they are the only source of error in our models. If we had more data, we could fit an estimator (*e.g.*, using maximum likelihood) to the proportion of promoted alerts in heterogeneous groups, and use that to provide an estimate on how many alerts should be sampled from an alert group before we reach a predefined level of certainty that we will not obtain false negatives. We attempted to do this in this work (not shown), but the results were inconclusive due to the rarity of heterogeneous groups, *i.e.*, we did not have an acceptable standard deviation of the maximum likelihood estimate to provide a reasonable level of certainty. Having more data would alleviate this issue.

7 Conclusion

The models presented in this work (barring the naive model, provided as a baseline and motivator) all meet our mentioned goals; they have high true positive and true negative rates; they are fast to build and fast to apply for prediction; they are easy to understand; and they all save analysts time and resources. We show that, on real data, our best performing model (heterogeneous group detection) can attain 99.958% specificity and 99.145% sensitivity, while automatically labeling 99% of alerts within an alert group, translating to a 92.322% reduction in initial labeling workload. To offset the overhead incurred by introducing false positives in this case, it must take ≥ 4354 times as long to further act on a promoted alert, on average, than to perform the initial labeling. Even higher sensitivity and specificity can be attained by decreasing the amount of automatic labeling (increasing the amount of manual labeling). We provide multiple trade-off curves with respect to sensitivity, specificity, and workload reduction, allowing potential implementors of our models to weigh these trade-offs and align with their specific goals. Our models are simple to understand and implement, which facilitates trust in their predictions. As our models primarily work due to the nature of attacks themselves (*i.e.*, attacks come in surges and thus generate homogeneous alert groups), we believe that our models will be applicable outside of our data. We believe that alert triage is very important practically, but is under-researched in the academic community. We hope that this work will act as a catalyst to drive this domain of research forward.

References

[1] J. Goldfarb, "7 tips to improve 'signal-to-noise' in the soc," April 2014. [Online]. Available: http://www.darkreading.com/analyt-ics/7-tips-to-improve-signal-to-noise-in-the-soc/d/d-id/1204605

[2] M. Riley, B. Elgin, D. Lawrence, and C. Matlack, "Missed alarms and 40 million stolen credit card numbers: How target blew it," March 2014. [Online]. Available: http://www.bloomberg.com/news/articles/2014-03-13/target-missed-warnings-in-epic-hack-of-credit-card-data

[3] J. E. Doak, J. Ingram, J. Shelburg, J. Johnson, and B. R. Rohrer, "Active learning for alert triage," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 2. IEEE, 2013, pp. 34–39.

[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," *arXiv preprint arXiv:1602.04938*, 2016.

[5] E. Sober, "The principle of parsimony," *The British Journal for the Philosophy of Science*, vol. 32, no. 2, pp. 145–156, 1981.

[6] J. Vandekerckhove, D. Matzke, and E.-J. Wagenmakers, "Model comparison and the principle of parsimony," 2014.

[7] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, p. 15, 2012.

[8] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[9] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[10] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE, 2003, pp. 14–23.

[11] Y. Xie, *A spatiotemporal event correlation approach to computer security*. ProQuest, 2005.

[12] G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with ga-based feature selection," in *Proceedings of the 43rd annual Southeast regional conference-Volume 2*. ACM, 2005, pp. 136–141.

[13] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, "Using machine learning technliques to identify botnet traffic," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, 2006, pp. 967–974.

[14] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer, "Applying machine learning techniques for detection of malicious code in network traffic," in *Annual Conference on Artificial Intelligence*. Springer, 2007, pp. 44–50.

[15] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh, "Modeltracker: Redesigning performance analysis tools for machine learning," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 337–346.

[16] Y. Xie, S. Tang, Y. Xiang, and J. Hu, "Resisting web proxy-based http attacks by temporal and spatial locality behavior," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 7, pp. 1401–1410, 2013.

[17] K. Sequeira and M. Zaki, "Admit: anomaly-based data mining for intrusions," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 386–395.

[18] S. Venkataraman, *Traffic Analysis for Network Security Using Learning Theory and Streaming Algorithms.* ProQuest / UMI, 2011.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

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

[21] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "A comparison of decision tree ensemble creation techniques," *IEEE transactions on*

pattern analysis and machine intelligence, vol. 29, no. 1, 2007.

[22] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, 2012.

[23] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[24] N. Rubens, D. Kaplan, and M. Sugiyama, "Active learning in recommender systems," in *Recommender systems handbook*. Springer, 2011, pp. 735–767.