SAND2013-6892C

Active Learning for Alert Triage

Justin E. Doak (J.D.), Joe Ingram, Jeffery Shelburg, Joshua Johnson, Brandon R. Rohrer
Sandia National Laboratories
Albuquerque, NM 87185, USA
Email: {jedoak, jbingra, jsshelb, jajohn}@sandia.gov, brohrer@gmail.com

***Abstract*—In the cyber security operations of a typical organization, data from multiple sources are monitored, and when certain conditions in the data are met, an alert is generated in an alert management system. Analysts inspect these alerts to decide if any deserve promotion to an event requiring further scrutiny. This triage process is manual, time-consuming, and detracts from the in-depth investigation of events. We investigate the use of supervised machine learning to automatically prioritize these alerts. In particular, we utilize active learning to make efficient use of the pool of unlabeled alerts, thereby improving the performance of our ranking models over passive learning. We demonstrate the effectiveness of active learning on a large, real-world dataset of cyber security alerts.**

I. Introduction

Most organizations have an operational cyber security group that is responsible for monitoring their networks, including preventing and responding to attacks. These groups are typically overwhelmed by the sheer volume of alerts and the asymmetric nature of cyber defense (i.e., cyber defenders must protect everything, but the attackers only need to find a single vulnerability and corresponding exploit). All of these groups would benefit from a system that helps prioritize alerts, whether those alerts are generated internally from sensors or externally via tips. Crucial alerts would not be overlooked, analysts would have more time for in-depth investigations, and the time gap between event and response might be reduced thereby potentially mitigating the impact.

Our solution utilizes supervised machine learning to provide a prioritized list of alerts to cyber analysts. A practical problem with supervised machine learning is obtaining labeled training data, which can be expensive in time and cost. By employing active machine learning, we only query the analysts for explicit labels on those alerts that are predicted to provide the most improvement in the output of the learned models. In practice, these methods typically require fewer labeled training examples than passive learning (i.e., randomly selecting instances for labeling) and are typically shown to be just as or even more accurate.

In this work, we investigate the performance of a simple active learning strategy, uncertainty sampling, on real-world cyber security alerts. We demonstrate that active learning vastly outperforms passive learning and requires fewer labels to approach the performance of a classifier trained on the fully-labeled dataset.

II. Related Work

Empirical and theoretical evidence indicates that active learning can outperform passive learning on a variety of learning tasks [16]. However, our literature review suggests that, while the field of active learning is well established in terms of its theoretical underpinnings, the application of active learning to real-world problems is in its infancy. In fact, we were only able to find a relatively small number of papers that employed active learning in cyber security settings.

One problem that has been noted when attempting to apply machine learning to cyber security problems is the lack of publicly available datasets. Many papers use the KDD-CUP'99 dataset [17], which has many documented issues [19]. One of these issues is that the high percentage of redundant records (78% and 75% in the training and test sets, respectively) biases the models towards the frequent records, which hinders the models generalization performance on unseen or infrequent records. In addition, the dataset does not present enough of a challenge for the models as even the worst model tested attained 86% accuracy. (The paper describes a new version of this dataset, NSL-KDD, that addresses both of these issues.) Another issue not mentioned in the paper is simply the age of the dataset meaning that many of the attacks are simply no longer relevant.

A real-world application of active learning to network traffic classification, anomaly detection, and malware detection is Aladin [18]. In Aladin, label queries are made based on two types of "interesting traffic": (1) instances that do not fit the pre-existing model (to discover new categories) and (2) instances that cannot be classified with high certainty (to improve classification accuracy). Results show that employing active learning greatly reduces the number of queries required to reach an acceptable level of accuracy and coverage. Furthermore, a new trojan was discovered in a real corporate network log containing roughly 13 million entries that rule-based methods had missed.

In [15], a team at Google utilized active learning to help detect real-world, adversarial advertisements. While experts traditionally discover new types of "bad ads" and emerging trends manually, active learning was used to rapidly build models to detect these new types of ads. Results show that only a few dozen label queries using active learning may be needed to build sufficiently accurate models.

In [2], MITRE explains their method of detecting scanning and probing attacks[1]. In their experience, MITRE notes that most alerts are just noise (i.e., false positives), and it is therefore imperative to reduce and prioritize alerts to more efficiently use analysts' time. Their approach includes selecting relevant features, aggregating alerts into episodes, filtering

[1] Currently, most networds are subjected to phishing attacks, drive-by downloads (i.e., legitimate, but compromised websites), and other types of intrusion attempts. Very little attention is paid to scans and probes as these can be automatically blocked.

episodes into scans, and ranking scans by severity. Furthermore, domain knowledge is used in classification to reduce the number of false alarms and clustering is employed to detect anomalies.

III. Data Collection

As previously noted, an alert is generated from monitoring real cyber security data and is then logged to a centralized alert management system (AMS). In the following sections, we briefly describe how we extract potentially relevant features (a vector of attributes that are consistent across all alerts) and how we obtained an initial dataset of labeled alerts based on the lifecycle of an alert.

A. Feature Extraction

An alert in the AMS contains both the raw text of the alert and some features extracted via regular expressions from the raw text (i.e., metadata). The raw portion of an alert does not have a standard structure, which has made feature extraction challenging. The metadata contains specific information about why the alert was created and why it may be worthy of further review by an analyst. In addition, since the alerts are largely composed of textual data, natural language processing techniques are used to generate many of the features. For example, named-entity recognition (NER) is used to extract different entities (e.g., filenames) from the alerts. Different techniques are then applied to the entities in order to derive potentially relevant features. For example, an alert could be correlated with other alerts containing specific entities in order to determine the fraction of related alerts that were malicious or benign.

Additionally, latent Dirichlet allocation [1] is used to model the inherent "topics" in the raw alert text. It is a generative, probabilistic model based on the notion that most documents are comprised of multiple "latent" topics. In a corpus of documents, LDA extracts a pre-specified number of topics, but each document in that corpus will differ in terms of its focus on the various topics. It utilizes the bag-of-words model and assumes that each word's usage can be attributed to one of the document's topics. The algorithm uses Bayesian inference (e.g., variational Bayes) in order to learn these latent topics.

A modification of the original algorithm [8] allows the model to be built from small, random subsamples of the corpus, which means the entire corpus does not need to be loaded in memory during the learning phase. We used the authors' implementation[2] to build the model and extract topics. The number of topics was chosen arbitrarily based on the perceived number of distinct alert types. The vocabulary that was used to build the model was based on the output of NER.

We are currently building our models using all of the extracted features and are not explicitly performing any kind of feature selection. In some cases (e.g., Random Forest), the model itself selects the features it deems most useful for the task at hand.

B. Implicit Label Extraction

Feedback from analysts will be both explicit (i.e., analyst labels) and implicit (i.e., alert life-cycle) and will be utilized to update models. (It is likely that we will give a higher weight to alerts that have been explicitly labeled.) To understand the alert life-cycle, consider that an alert's status is "open" on creation, but changes as an analyst triages and makes decisions about the importance of the alert. The current status of an alert, along with other information, allows us to infer an implicit label for it. These implicit labels will augment the explicit labels obtained from analysts and have also allowed us to build models before explicit labels are obtained.

In this paragraph, we discuss the implicit labeling of alerts, from the least important alerts to the most important. For example, if an alert is closed by an analyst and never promoted to an event, we label it as a "false positive". The "open not viewed", "open viewed", and "revisit" categories of alerts represent alerts that have not been closed as false positives, nor have they been promoted to an event. Hence, it is these alerts that our model(s) ranks. "promoted false positive" represents alerts that were promoted to an event, but then marked as a false positive. The "promoted" label is given to alerts that were promoted to an event and were not subsequently marked as a false positive or an incident. The final category of alert is "incident" and indicates events that must be reported.

For our initial results, we mapped all of these labels into "closed" and "promoted" classes (e.g., "false positive" was mapped to "closed" and "promoted false positive", "promoted", and "incident" were all mapped to "promoted"). This mapping allows us to frame the problem as binary classification while still accurately simulating the triage process.

IV. Supervised Learning

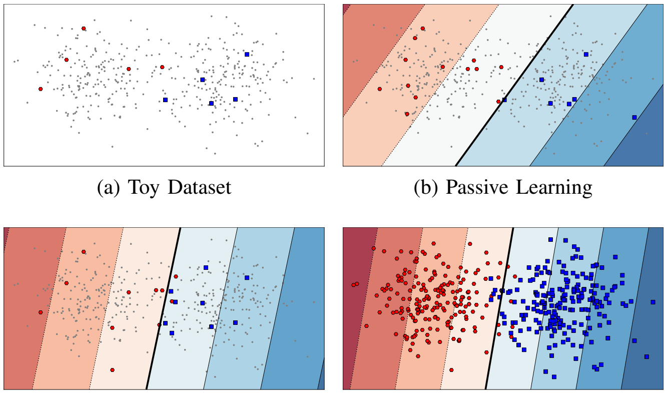
In this application, supervised learning is used to build a ranking model that accurately prioritizes open alerts. (Note that many classification models can be adapted to ranking applications, provided that the probabilities of class membership are provided.) Before we can select an appropriate supervised learning algorithm, we must evaluate the ranking performance of the candidate models. A wide variety of supervised learning methods, each with their own set of advantages and disadvantages, were evaluated during the model selection process.

While there exist many ways to logically categorize these supervised learning methods, categorizing them as *linear* or *nonlinear* is convenient because of its simplicity and descriptive power. A *linear* supervised learning method will yield a model with a decision surface that can be described using a linear combination of input features, whereas a *nonlinear* supervised learning method will do so using a nonlinear combination of input features. Brief descriptions of each supervised learning method evaluated are given in their respective *linear* or *nonlinear* subsection below.

A. Linear Methods

1) Linear Support Vector Machine (SVM): An SVM represents data as points in input feature space and finds a hyperplane that divides the classes with as wide a gap as possible by solving an optimization problem [4] utilizing Platt's Sequential Minimal Optimization algorithm [12].

[2] http://www.cs.princeton.edu/~blei/topicmodeling.html


(a) Toy Dataset

(b) Passive Learning

(c) Active Learning

(d) Full Learning

Fig. 1. Active vs. Passive Learning on a Toy Dataset

2) Logistic Regression: This technique models class probabilities as a function of input features using a logistic function.

3) Linear Discriminant Analysis (LDA): LDA attempts to find a linear combination of input features that best distinguishes between classes using the log of the likelihood ratios under the assumptions that classes are normally distributed and class covariances are identical [7].

4) Naïve Bayes: This is a probabilistic classifier that utilizes Bayes' theorem under the assumption that input features are independent [13].

B. Nonlinear Methods

1) SVM with Radial Basis Function (RBF) Kernel: This model is the same as a Linear SVM except that an RBF is used to measure distances, resulting in a nonlinear classification boundary [20].

2) k-Nearest Neighbors (KNN): KNN classifies an instance based on the class labels of its *k* nearest points in feature space.

3) Quadratic Discriminant Analysis (QDA): This algorithm is the same as LDA except it does not assume class covariances are identical and therefore produces a quadratic combination of input features.

4) Multilayer Perceptron (MLP): This model is comprised of a network of simple processing elements (neurons). Data features are presented to the input layer where neuron activations are then fed forward in the network through the hidden and output layers to yield a final classification. When training, error values are propogated backwards through the network to train the adaptive weights between neuron layers.

5) Random Forest: This is an ensemble method that constructs many bootstrap aggregated (bagged) decision trees that split on a random, fixed-sized subset of input features at each node in each tree [3].

V. Active Learning

Traditionally, most supervised learning problems assume that labels for every data point are available to the learning algorithm. However, in some real-world applications, there are only a small fraction of labeled samples and the vast majority of samples are unlabeled. In addition, the process for obtaining labels can be costly in both time and effort, as it generally requires a trained human annotator. Active learning attempts to maximize the utility of labeled data by querying an oracle for labels on only a carefully selected set of unlabeled instances. These instances are the ones that, once labeled, are predicted to most improve the output of the learned models. The principal idea behind active learning is that a supervised learner can perform as well or better than passive learning with less training data if it is allowed to choose the data from which it learns [16].

Fig. 1 (generated using `matplotlib` [9]) illustrates this point by showing a simple two-dimensional dataset in which the learner is trying to discriminate between *red hexagons* and *blue squares*. Fig. 1a shows the initial dataset, where the labeled points are marked with the appropriate colored markers and the unlabeled points are represented as small gray dots. Fig. 1b represents the learned decision boundary after randomly (passively) selecting ten new points to label. Fig. 1c depicts the learned decision boundary after ten iterations of active learning, in which the active learner selects the point closest to the current decision boundary for labeling. Fig. 1d represents the learned decision boundary if all of the data points were labeled. It is easy to see that the actively learned decision boundary is much closer to the decision boundary learned on the fully labeled dataset. Although this simple example demonstrates the efficacy of using active learning, the question now becomes: how is the best instance(s) to label selected?

A. Query Frameworks

Before discussing how individual instances are selected for labeling, we briefly describe some common settings in which active learning is typically applied. In general, there are three different frameworks for active learning [16]:

1) Query Synthesis: In this setting, a learner is allowed to request labels for any data point in the input space, including points that have been synthesized. The only assumption in this setting is that the definition of the input space (i.e., the features and their associated ranges) is known. However, this framework does not typically work well for problems in which points must be labeled by a human oracle, as synthetically generating points may result in instances that are nonsensical (e.g., in character recognition, it might produce characters that are difficult for a human to distinguish).

2) Selective Sampling: In this setting, each instance arrives sequentially and the learner must decide at that moment whether to query for a label or discard. The decision whether to query or discard an instance can be framed in many ways, but typically involves either defining a utility measure for making a biased random decision or computing an explicit region of uncertainty and only querying instances that fall within it. It is typically only used in streaming settings.

3) Pool-based Sampling: In this setting, there is a set of labeled instances and a pool of unlabeled instances. The learner is allowed to look at all instances in the unlabeled pool in order to select the optimal query, which is typically done in a greedy manner. (A greedy approach can create issues, such as querying instances that are too similar to other instances that have already been queried. There are techniques for addressing problems such as this.) This setting appears to be more common in practical applications.

B. Query Strategies

Once the appropriate active learning framework is selected, the next step involves choosing the appropriate strategy for identifying the points to be labeled by the oracle.

1) Random Sampling: The simplest approach for selecting which examples to label is to sample them uniformly at random. However, since no information about the input space or current supervised model is used to select points, we refer to this strategy as *passive* learning. Random sampling is not technically a form of active learning, but it is often used as a baseline for comparison. It should also be pointed out that there are datasets for which random learning outpeforms active learning [16].

2) Uncertainty Sampling: In this approach, first introduced by Lewis and Gale [10], unlabeled instances that the model is least certain how to classify are selected for labeling. Uncertainty sampling is based on the idea that obtaining labels for instances about which the current model is most uncertain will yield more information than instances in which the model is highly confident (because these are most likely correct).

Measuring this uncertainty involves estimating the distance from a particular example to the current model's decision boundary. Although some classifiers explicitly output the distance to the decision boundary (e.g., SVMs), explicit uncertainty measures must be defined for other classifiers (e.g., probabilistic classifiers). The most common measures of uncertainty for probabilistic classifiers are [16]:

- **Least Confident** – query the instance whose predicted output is the least confident:

  $$\underset{x}{\text{argmax}}\, 1 - P_\theta(\hat{y}|x)$$

  where $P_\theta(\cdot)$ represents the posterior distribution of the model θ (i.e., the probability that the label is y, given input x), and $\hat{y}$ is the prediction with the highest posterior probability for the given input x.
- **Margin** – query the instance with the smallest margin (difference betwee the two most likely predictions):

  $$\underset{x}{\text{argmax}}\ [P_\theta(\hat{y_2}|x) - P_\theta(\hat{y_1}|x)]$$

  where $\hat{y_1}$ and $\hat{y_2}$ represent the first and second highest posterior probabilities given the input x.
- **Entropy** – a measure of a variable's average information content. The instance with the highest entropy is queried:

  $$\underset{x}{\text{argmax}}\ -\sum_y P_\theta(y|x)\ \log\ P_\theta(y|x)$$

  which represents the entropy of the model's posterior distribution for the given input x.

In learning problems with only two classes, these measures are equivalent. However, in multiclass learning problems, the least confident and margin-based uncertainty measures tend to select points that will reduce classification error. For example, they might select an instance that would help the model better discriminate among specific classes. On the other hand, the entropy measure will attempt to minimize expected log-loss, which means it will select points where the current model's posterior distribution is most uncertain (i.e., closest to uniform).

Uncertainty sampling is very easy to implement, since any classification algorithm that outputs an appropriate distance metric from a point to the decision boundary can be used as a "black box." However, the initial model is often trained with very little data, and subsequent data might become biased by the sampling strategy [16].

3) Other Strategies: Uncertainty sampling is not the only method for selecting which point to query. In fact, there are a plethora of methods that have been developed in the active learning literature, which can be broadly categorized into hypothesis-space search, expected-error or variance reduction, and exploiting structure in data [16]. However, since we have currently only explored uncertainty sampling, we omit discussion of these other methods in the interest of brevity.

VI. Experimental Results

Initial experiments were run on real-world alert data consisting of 8905 alerts, 1436 (approximately 16%) of which were promoted to an event. For implementations of the various algorithms, with the exception of the MLP, a machine learning library developed in Python called `scikit-learn` [11] was used. Since a MLP implementation was not included in `scikit-learn`, another Python library called `PyBrain` [14] was used for the MLP.

A. Model Evaluation

Model evaluation consists of techniques such as cross-validation that attempt to maximize the value of the labeled data. They achieve this by accurately estimating the generalization performance of a classification algorithm without using a separately held-out test set. For example, 10-fold cross-validation splits the data set into ten "folds". The first evaluation uses nine of the folds to train the model, and then evaluates on the remaining fold. When all ten of the folds have been used as the testing set, the performance across all of the folds is averaged.

Each evaluation (a total of ten in the 10-fold cross validation example) relies on a metric that scores the utility of the model. These metrics can be categorized as classification, probabilistic, or ranking depending on the type of model a metric most effectively evaluates [6]. For our application, we are focusing on ranking metrics because our end goal is to prioritize alerts for analysts. An example of a metric that may be effective at evaluating ranking models is the Area Under the Receiver Operating Characteristic Curve [5].

In order to measure the classification performance of each supervised learning method on our problem, the macro average arithmetic metric [6] was used where the accuracies of predicting each class individually are averaged over all classes. For clarity, this metric will henceforth be referred to

TABLE I. Baseline Model Comparison

	Method	CAA
Linear	LDA	0.774 (0.019)
	Naïve Bayes	0.684 (0.016)
	Linear SVM	0.585 (0.015)
	Logistic Regression	0.556 (0.014)
Nonlinear	Random Forest	0.814 (0.020)
	QDA	0.753 (0.074)
	MLP	0.560 (0.021)
	SVM w/ RBF	0.516 (0.007)
	*k*NN	0.457 (0.014)

as class-averaged accuracy (CAA). This metric is used instead of overall classification accuracy due to the presence of skew in the class distribution, which may bias certain classifiers towards predicting the majority class (e.g., *closed* alerts) often to the detriment of the minority class (e.g., *promoted* alerts). We mitigate this tendency to neglect the minority class by choosing the supervised learning method that maximizes CAA.

B. Baseline Performance

To select an appropriate supervised learning method, baseline performance for each method is needed. This performance was obtained by performing three runs of 10-fold stratified cross-validation and calculating the CAA for each of the 30 total folds as well as the overall average and standard deviation for each method. These values are shown in TABLE I (standard deviations in parentheses). The random forest method (using 100 base decision trees) was determined to be the best model based on average CAA. Furthermore, a Wilcoxon signed-rank test [21] with $\alpha = 0.05$ determined that the differences between the random forest and all other evaluated methods were statistically significant.

C. Active Learning Performance

In order to evaluate the performance of active learning on the alert data, a slight variation of 10-fold stratified cross-validation was used. The ten stratified folds were selected, and then, for each iteration, 10% of the data was randomly sampled (without replacement) from the training folds to build an initial model. Next, each of the active learning strategies were allowed to select from the remaining 90% of the data in the training fold. The model was then evaluated against the held-out test fold. In order to reduce the variance associated with the initial 10% subsampling, the process was repeated ten times for every fold of cross-validation.

Since the entire unlabeled dataset is available, we investigate the performance of pool-based sampling described in Section V-A. The model was allowed to select 50 instances from the pool at a time to be added to the labeled dataset, and then the model was retrained and the selection process repeated.

Fig. 2 shows the CAA of the random forest model as a function of the number of queried labels for uncertainty sampling (active) and random sampling (passive). The plots represent the average over the 100 iterations (10 folds × 10 iterations of randomly subsampling 10% of the training data to build an initial model), with error bars at one standard error.

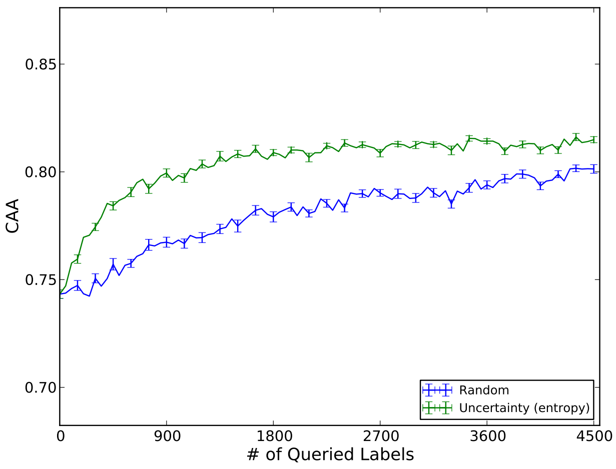


Fig. 2. Active vs. Passive Learning on Alert Data

As can be seen in Fig. 2, active learning vastly outperforms passive learning, and it approaches the baseline performance (see TABLE I) of the random forest model trained on the fully-labeled dataset after approximately 1500 queries. Given that the initial model was trained on 10% of the data in the training folds and the queried data[3] represents approximately another 20%[4] of the data, active learning is able to achieve the same performance with approximately a 70% reduction in labeled data for this dataset.

VII. Deployment to Enterprise Security

The described active learning-based ranking model(s) will be integrated into an AMS to automatically prioritize alerts for analysts. This will allow the analysts to focus their triaging efforts on alerts most likely to be promoted to events and may also increase the time available for in-depth investigations of events. As new alerts are created, the model will predict their labels and insert them into the tracking system in the appropriate location relative to all the other open alerts. On a nightly basis, the model(s) will be rebuilt using all available labeled alerts, including the labels (both explicit and implicit) obtained the previous day. Then, all open alerts will be relabeled in batch-mode. This relabeling is performed during off-peak times so as not to cause unnecessary load when the AMS is being heavily used by analysts.

The AMS allows analysts to modify the contents of an alert. For example, if an analyst discovers new information during the triaging process, he or she may append this information to the contents of the alert. When any open alert is modified in such a way as to cause a change in value for any of its features, features will again be extracted, the model will predict a new label, and the alert will be moved to the appropriate location given its new priority.

Multiple analysts have independently requested that we periodically project closed alerts onto our newly-built model(s).

[3] In this context, we mean the queried data necessary to approach the accuracy of the model trained on all nine folds.

[4] The training folds consist of roughly 90% of the data, or approximately 8000 alerts.

By reprioritizing closed alerts, it is possible that the current model, with the advantage of new information, may deem some of the closed alerts as likely to be promoted. This may prompt an analyst to revisit these closed alerts to see if actual events were missed and begin their respective investigations.

VIII. Conclusion

In the operational cyber security group of a typical organization, data from multiple sources are monitored, and when certain conditions in the data are met, an alert is generated in an AMS. Analysts must inspect these alerts to decide if any deserve promotion to an event, which is typically a very manual, time-consuming process. We have investigated the use of active learning to automatically prioritize alerts so that analysts can effeciently triage alerts, focus on in-depth investigations, and rapidly respond to potential threats. We have shown that, for our data, active learning results in a 70% reduction in the number of labels required to achieve the same performance as when using the fully-labeled dataset.

Acknowledgment

Funding for this work came from the Laboratory Directed Research and Development program at Sandia National Laboratories.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly-owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

The authors would like to thank Dustin Franklin, Nick Peterson, Todd Bruner, Roger Suppona, Beth Potts, Bill Hart, and Danny Dunlavy for useful discussions, insights, and contributions.

References

[1] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, p. 2003, 2003.

[2] E. Bloedorn, L. Talbot, and D. DeBarr, "Data Mining Applied to Intrusion Detection: MITRE Experiences," in *Machine Learning and Data Mining for Computer Security*, ser. Advanced Information and Knowledge Processing, M. Maloof, Ed. Springer London, 2006, pp. 65–88.

[3] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] C. Cortes and V. Vapnik, "Support Vector Machine," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[5] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[6] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An Experimental Comparison of Performance Measures for Classification," *Pattern Recogn. Lett.*, vol. 30, no. 1, pp. 27–38, Jan. 2009.

[7] R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[8] M. D. Hoffman, D. M. Blei, and F. Bach, "Online Learning for Latent Dirichlet Allocation," in *In NIPS*, 2010.

[9] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, May-Jun 2007.

[10] D. D. Lewis and W. A. Gale, "A Sequential Algorithm for Training Text Classifiers," in *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '94. New York, NY, USA: Springer-Verlag New York, Inc., 1994, pp. 3–12.

[11] 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.

[12] J. Platt *et al.*, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," 1998.

[13] I. Rish, "An Empirical Study of the Naïve Bayes Classifier," in *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[14] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *Journal of Machine Learning Research*, 2010.

[15] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou, "Detecting Adversarial Advertisements in the Wild," in *Proceedings of the 17th ACM SIGKDD International Conference on Data Mining and Knowledge Discovery*, 2011.

[16] B. Settles, *Active Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[17] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *In Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[18] J. W. Stokes, J. C. Platt, J. Kravis, and M. Shilman, "Aladin: Active Learning of Anomalies to Detect Intrusions," *Technique Report. Microsoft Network Security Redmond, WA*, vol. 98052, 2008.

[19] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[20] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A Primer on Kernel Methods," *Kernel Methods in Computational Biology*, pp. 35–70, 2004.

[21] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.