Building Intrusion Detection System using Machine learning

Using Active Learning and Upper Confidence Bound to detect Intrusions

Krishna Chaitanya Gopaluni kgopalun@sfu.ca

ABSTRACT

Traditionally, intrusion detection systems have relied on rule based signatures to detect intrusion in networks. Rule based IDS systems generate too many false positives and require a human to define rules and signatures by hand. With the web traffic increasing at a fast pace and threats along with it, such a manual and noisy method is not scalable. With increasing number of false positives, organizations will have to deploy even greater task force for handling network intrusions. Additionally, the nature of threats change everyday and hence the signature based systems can't detect zero day attacks. Machine learning algorithms on the other hand do not memorize the rules but learn discriminative features to identify anomalies. We have utilized 1999 KDD intrusion detection dataset for this task as it is widely used to benchmark anomaly based intrusion detection system by researchers. We modelled the problem as a binary classification problem and tested the well known XGBoost classifier to detect intrusion classes as a baseline model. In this project, our endeavour is to improve upon this baseline.

We ended up using active learning for this problem as in the real world, it will not be known if the network connection is an attack. Thus the data will be unlabelled in nature. Active learning is designed to work with sparsely labelled data and it also enhanced model performance and also reduced the number of iterations to reach max performance by querying only the most uncertain samples for labels. We also utilized the upper confidence bound algorithm, a classic solution to the reinforcement learning problem classed multi-armed bandits, to select the sampling technique dynamically for each iteration of active learning.

1 INTRODUCTION

Intrusion detection systems come in two flavours - rule based and anomaly based. Rule based intrusion detection systems require domain experts to craft signatures by hand and are generally well understood. But rule based IDS systems are not adaptable and can't detect zero day attacks. As with any intrusion detection system, the overall aim is to reduce false positives and detect maximum amount of attacks. Anomaly based detection systems learn to categorize connections into attack or normal using the input features as deciding factors. We have used the famous KDD 19 which is a popular dataset among security researchers for benchmarking intrusion detection systems.

This dataset contains a wide variety of attacks and is used for auditing and research in the field of IDS. Also, the dataset has well documented and well defined features. The dataset was collected and prepared by DARPA in 1998 as a part of Intrusion Detection Evaluation Program at MIT Lincoln Labs. This dataset contains a wide variety of intrusions - containing both simulated and real world scenarios. Each observation in this dataset represents a TCP connection for a Local Area network. There are about 5 million observations in the training data. The test data is from entirely different distribution which makes the problem statement more realistic. In reality, the zero day attack does not match with the data that model trained on. Also, indifferent to the industry application, the deployed model, needs to be updated frequently (iterative or continuous training) causing fluctuations in model performance with imbalanced data.

This boils down to the idea of building generalized model that needs techniques like active learning as it is good at generalization with stream of new data coming in (which causes an imbalance of classes) for frequent model updates.

The dataset contains four main attack types - Denial of Service, Remote Root Access, Local Root Access and reconnaissance attacks. In denial of service attacks, computing or memory resource become too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. User to Root Attack (U2R) is a class of exploit which gains access to user account perhaps by sniffing passwords, a dictionary attack, or social engineering and the exploit vulnerability to gain root access. Remote to Local Attacks (R2L) exploits vulnerability to gain local access as a user of the machine through packets over a network. In probing attack, the attackers attempt to gather information about a network of computers for the apparent purpose of circumventing its security controls.

These four categories contain many sub types of attacks. The distribution of training and test datasets are different to simulate real world scenarios. The training dataset contains a total of 24 attack types. The test dataset contains all these attack types and additionally it contains 14 attack types not present in the test dataset. The main type and subtypes of attacks have been summarized in Table 1.

The 14 types of attack that are only present in the test dataset are: apache2, worm, xlock, xsnoop, xterm, httptunnel, processtable, ps, saint, sendmail, snmpgetattack, snmpguess, sqlattack, and udpstorm.

The dataset contains 3 types of features - basic features, content features and traffic features. The basic features encapsulates all the attributes that can be extracted from a TCP/IP connection. Most of these features leading to an implicit delay in detection. The dataset also contains time based traffic features - connections having the same host as current connection in the past two seconds and connections having the same service in the past 2 seconds as the current connection. There are window features - a window of 100 connections or a window of all connections in past 100 seconds - to capture the temporal nature of cyber attacks. Apart from traffic and window features, the dataset also contains content features like number of failed login attempts. The content features contain the information about the what the connection is actually trying to do.

ATTACK CLASS	ATTACK TYPE
Probing Attack	ipsweep, nmap, portsweep, satan
DoS	back, land, neptune, pod, smurf, teardrop, apache2, processtable, sendmail, udpstorm
R2L	warezmaster, warezclient, spy, phf, multihop, imap, guess_passwd, ftp_write, xlock, xsnoop, httptunnel, snmpgetattack
U2R	buffer_overflow, loadmodule,perl, rootkit, sqlattack, ps, xterm

Table 1: **Attack Class and Types in the dataset**

2 LITERATURE SURVEY[3] [5]

Support Vector Machines have high training times and memory requirements which make them a poor fit for scalability. This problem is exacerbated in domains like security testing where the data generated every second is huge. There have been many techniques proposed over the years to improve upon the scores achieved on IDS datasets using SVM models for the very same reason. We will have a brief tour of these approaches in this section.

Multiple PCA based techniques have been proposed to deal with high training times of Support Vector Machines over the years. Under normal circumstances, PCA will select the principal components which explain the maximum amount of variance in the data. But since these components may not be important in classification of attack and normal class, the authors in [8] chose select principal components with higher Fisher Discriminant Ratios to reduce the training time for SVMs. In [9], the authors used kernel PCA to extract non-linear information from the KDD 99 dataset to increase the detection rate of attack class by 5 percent. In [10], the authors used a combination of information gain and correlation based feature selection to narrow down the KDD 99 dataset to only 10 features for the SVM to train on. Information gain is calculated between each feature and the class. Features are ranked in decreasing order of information gain values and top k features are selected for training purposes. The authors in [10] also used the process of discretization to convert continuous variables into discrete ones as well as using correlation based feature selection to reduce SVM training times. In [11], the authors represented each observation of the KDD 99 dataset using only two distances - distance between the observation and the center of the cluster to which it belongs after the clustering process and distance between the observation and it's nearest neighbour in the same cluster. This representation greatly reduces the training time and performs better than SVM.

In [12], the authors achieved an accuracy of 96 percent by averaging k-best Bayesian network classifier with significantly reduced training times. In [13], the authors used beta profiling to reduce the number of observations and achieved an accuracy of 98.66 percent using an online flavour of the extreme learning machines. In [14], the authors used Decision Trees (CART) for feature selection, followed by feature transformation using DWT (discrete wavelet transform) and using SVM to as the final classifier in the pipeline to achieve an accuracy of 95.5 percent on the KDD 99 dataset.

In this project, we go above and beyond these techniques to do both feature selection and classification using tree based XGBoost model and ultimately utilize active learning techniques to conquer real world challenges like sparsely labelled network data and a continuously incoming stream of potential attack connections. Using the above mentioned techniques, we observe a fairly substantial improvement in both accuracy and precision values while also taking care of the use case when either the data is unlabelled or there is an incoming stream of continuous data or both.

3 IMPLEMENTED METHODS

The following sections contain all the methods that we have implemented to tackle this classification problem. We modelled the problem as a binary classification problem - either a connection belong to the normal connection class or attack class. We think this is a fair assumption to make as we typically only interested in knowing whether an attack is malicious. To reduce false positives, we also have implemented active learning techniques to query the human for labels if the model is unsure about the prediction. This should in theory lead to less false positives as a human is added in the loop if the model is unsure about the class of the connection. This method of leverage and combine both human as well as machine intelligence into decision making should improve the evaluation scores.

3.1 Evaluation Metrics

As discussed, the KDD 99 dataset contains class imbalance with 1 million records labelled as “normal connection” and about 4 million records labelled as “attack class”. Because of this imbalance, any model that outputs all connections as “attack” without any regard to input features will have an accuracy of 80%. Hence accuracy is an invalid metric to use for this problem.

To overcome class imbalance problem, we will select our classification metrics to be such that they are immune to imbalanced dataset. Hence we have selected Precision to be our primary evaluation criteria and accuracy to be our secondary evaluation criteria.

Precision or positive predictive value is defined as the number of true positives divided by the sum of false positives and true positives. False positives are the observations which our model identified as “attack connection” and they were actually “normal connections”. We want to minimize false positives and hence we will keep track of precision values for various machine learning models and algorithms. We will track both - Precision for attack class as well as Precision for “normal” class and report both these values for our experiments.

3.2 XGBoost as baseline model [1]

XGBoost is a modified flavour of the Gradient Boosting Machines. It is highly scalable, fast to train and battle tested. XGBoost is so versatile that it can be used for regression, classification as well as ranking problems. Due to its superior model performance and execution speed, we selected XGBoost to be our baseline classification model.

XGBoost is an ensemble algorithm that seeks to combine a series of weak learners to build a strong learner. By building models on top of each other iteratively, the residuals of previous learners are fed to the next learners and weightage assigned to incorrectly classified observations is increased. Thus in a serial fashion, incorrectly classified observations are minimized.

The reason why XGBoost outperforms not only the GBM model but also other classification models is due to some system and algorithmic enhancements applied to GBM which create the XGBoost algorithm. XGBoost is highly parallelized with the ability to use all the cores of CPU for training weak learners in a parallel fashion. XGBoost also comes inbuilt with an early stopping criteria and tree pruning built in which greatly help in controlling overfitting. Also, XGBoost penalizes models with high variance by using L2 regularization which is inbuilt. XGBoost also has built in feature sub sampling where each weak learner is not built on all the features but on a set of randomly selected features. This is very similar to random forest algorithm and hence XGBoost combines the best of both worlds - bagging as well as boosting. Inbuilt cache awareness and automatic handling of sparsity of data also play a significant factor in high performance of XGBoost classifiers. All these qualities make XGBoost an ideal candidate as our baseline classifier.

Throughout the project, our endeavour will be to improve upon the scores or evaluation metrics as reported by XGBoost classifier.

3.3 Active learning in action [1]

As discussed in the previous section we wanted to deal with model training issues that occurs when the stream of new training data creates imbalanced classes and lack of abundant labeled data causing weak model performance. In reality it’s a painful and costly process to obtain labels and one of the reasons why the ideas like mechanical turks turns out to be successful. In order to increase the model performance with least labeled data points as possible we are opting active learning strategy.

Active learning is an iterative training where in each iteration model picks uncertain samples from the pool of unlabeled data points. Intuition behind this idea is that over the iterations model tries to be confident on the information that is not certain about which is not the case without active learning.

We have implemented pool based sampling methods to pick uncertain samples. Entropy, margin and least confidence are some of the strategies used for sampling. Experiments and dataset distribution decides the good strategy to go for. Active learning can be implemented on any machine learning algorithm. Indifferent to ML algorithm and sampling strategy, the typical process of active learning looks like this:

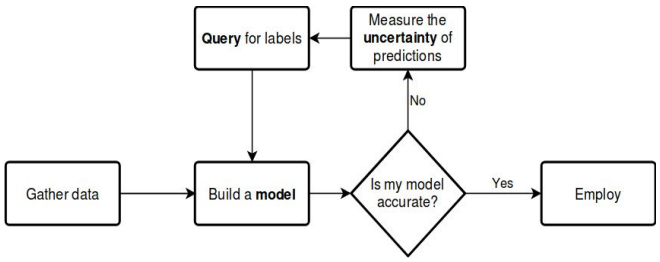

Figure 1: **A typical active learning algorithm flowchart**

- Assume all the available labeled data as split of labeled and unlabeled data points.
- Fit the model on initial set of labeled samples. And use this weak model query for the uncertain samples from the assumed to be unlabeled set of samples.
- Append them to the labeled samples and trash the current model. Fit new model on the updated labeled data.
- Repeat this process until unlabeled samples are over.

We have implemented the code for active learning and pool based sampling techniques from scratch by following the information form the paper [1]

3.3.1 Margin Sampling*. In the case of binary classification,* the least confidence method says the closer the probability of the label is to 0.5 (05. split b/w 2 labels), the least confident the model is of its predictions. This sample is picked for labeling but only with 0.5 probability. If the model does so, there could be a loss of information from other label predictions. In the intrusion detection, there could be a chance of the model misclassifying a data point as not an attack with only half of the chance. This can create more false positives. In order to deal with this issue, we use margin sampling for querying the uncertain data points.

Margin sampling selects a data point with least valued difference of the last two predicted probabilities with maximum value. By considering the last two probable labels for a given datapoint, there could be no information loss.

$$x_M^* = \underset{x}{argmin}\ P_\theta\left(\hat{y}_1|x\right) - P_\theta\left(\hat{y}_2|x\right) \quad \text{--(1)}$$

where x is the most informative instance under algorithm M and $\hat{y}_1$ and $\hat{y}_2$ are the first and second most probable class labels. Notice how we take the argmin, as the most informative sample is the one with the smallest difference between the top two label probabilities. Instances with a small margin are more ambiguous, hence knowing their label will help the model improve its discrimination capacity on those two classes.

3.3.2 Entropy Sampling*.* Entropy sampling is one other technique that identifies most uncertain samples using entropy value obtained by the predicted probabilities of a datapoint. The magnitude if entropy value explains how informative a sample is. If the sample is more informative it means the model is uncertain about the sample and the probabilities of predicted labels of a data point might be spread across.

$$x_H^* = \underset{x}{argmax} - \sum_i P_\theta\left(\hat{y}_i|x\right) \log P_\theta\left(\hat{y}_i|x\right) \quad \text{--(2)}$$

3.3.3 Random Sampling.Random sampling is performed for the purpose of comparison with uncertain sampling techniques. As the name implies random sampling would not follow any stratesy except to query random samples in each iteration of active learning and helpful benchmark to compare other sampling techniques in this problem statement.

3.4 **Upper Confidence Bound (UCB)** [2]

Unlike generally explaining UCB, which is a solution for classic reinforcement learning problem called multi armed bandit, we are going to discuss how UCB can help us automatically choose its own uncertainty sampling strategy in every iteration.

Let's say we have 0 to i number of uncertainty sampling strategies (in our case there are 3: margin, entropy and random). For the first i iterations of the active learning training repeat each technique one after the other once. Store the computed accuracies for the pool of unlabeled data in a vector where each index represents a different strategy. Call this vector as running means vector $\bar{\mu}$.

Just before the next iteration starts, the UCB values for all the strategies are calculated using formula $\bar{\mu}_i + \sqrt{\frac{2\ ln(n)}{n_i}}$ and the strategy that has max value of UCB is chosen in the next iteration (equation (3)). The UCB is a greedy approach to maximize overall gain (maximize the UCB value) by exploration and exploitation trade off.

$$arg\ max \quad \bar{\mu}_i + \sqrt{\frac{C\ ln(n)}{n_i}} \quad \text{--(3)}$$

$\bar{\mu}_i$ *– running means of the accuracies of i th sampling technique*

n_i *– number of times the i th sampling is repeated over iterations*

n *– number of iterations completed until the current iteration*

C *– Tunable parameter, that decides the amount of UCB maximization*

The denominator of the above equation n_i, maintains the number of times each strategy has been explored over the past iterations. So, for the i th strategy as the iterations are proceeding:

- If the numerator value (i.e. **running mean of i th strategy + 2*log[current_no_of iterations]**) does not go up
- and if n_i for the same i th strategy is high

Then, the upper bound of that particular strategy will be reduced. And, in the next iteration this strategy has less chances of selection by ucb algorithm as the accuracy of the model is not improving.

After the training has ended, the plot of strategies selected in each iteration would give us the depiction of best uncertainty sampling technique that has been selected by UCB most of the time. Sometimes chances are all of the techniques equally bounded in every iteration. In this scenario we can choose a different model evaluation metric to calculate running means.

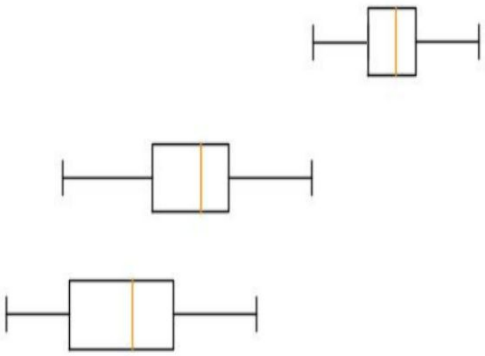

Figure 1: **Example interpretation of computed UCB values (boundaries of each box plot) when there are three competing alternatives. Size of boxplot represents number times the strategy repeated in the past iterations. Y axis is points the running mean of accuracies (mid point of yellow line).**

In Figure1, even though the running mean is high for the top most box (strategy 3), the bound was set to less because to give chance to explore other strategies that algorithm not sure of.

4 EXPERIMENTS & RESULTS

The baseline XGBoost model ended up giving 92.7% accuracy. The precision on the “attack” class was 90.88% and on the “normal” class was 70.99% for the baseline model. Observations belonging to “normal” class have less precision because the data is very imbalanced i.e out of 5 million datapoints approximately only 1 M of them belongs to the “normal” class.

We have dealt with this issue using active learning by slightly increasing the precision of “normal” class. Also, there is an increase in the test accuracies. 30 experiments were performed, 10 for each of margin, entropy and random sampling techniques. Each experiment has a shuffled data at different levels. The figure 1,2,3,4,5,7 are the results of active learning.

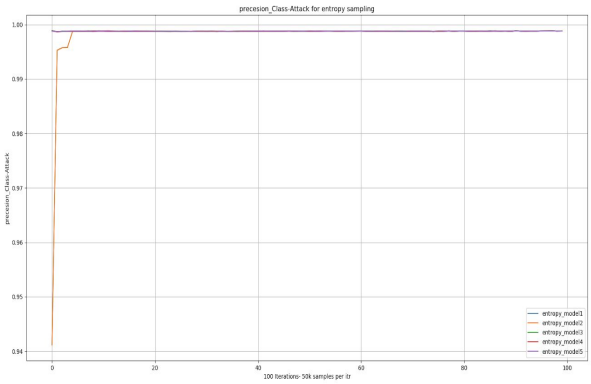

Figure 2: **Test Precision score for attack class using Entropy.[The precision has gone up from 90.88 to above 98% compared to the baseline model. Similar scores were generated with other sampling techniques.]**

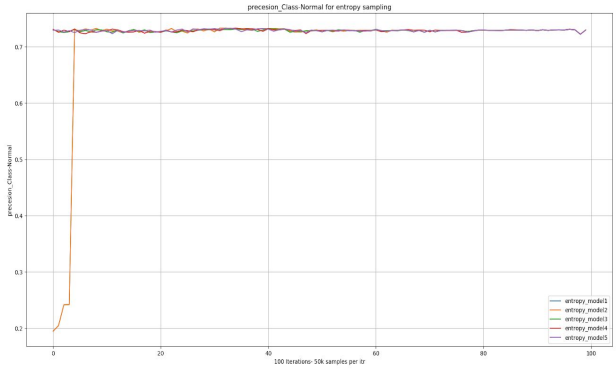

Figure 3: **Test Precision score for normal class using entropy sampling. [The precision has gone up from 70.99 to above 74% compared to the baseline model. Similar scores were generated with other sampling techniques]**

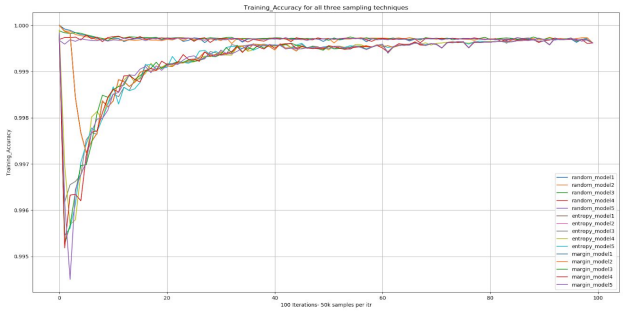

Figure 4: **Training accuracy score representing how well the model has learned from training data. [The set of lines that originates from the top left are multiple experiments of random sampling. Those originates**

from bottom left are different active learning techniques of multiple experiments. All of them are converging at same accuracy]

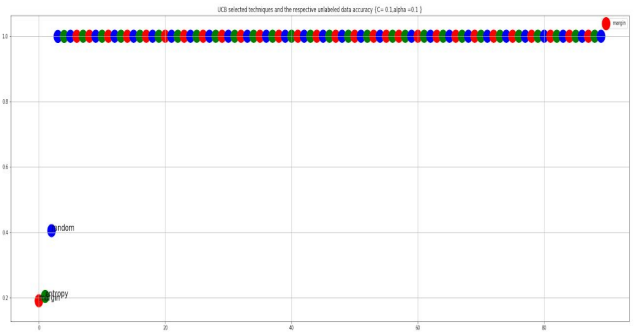

Figure 6: **Dynamic Selection of sampling by UCB. [UCB has choose all the strategies similarly but figure 5, has given difference b/w random and AL strategies]**

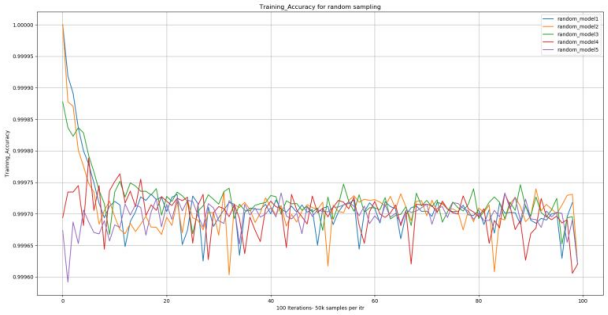

Figure 7: **Random Training accuracy score representing how well the model has learned from training data. [Even though AL and random sampling converge at the same point, the training process of random sampling has fluctuating performance and seems unreliable for deployment.]**

5 CONCLUSION

Compared to the baseline XGBoost classifier, using active learning techniques we have improved the precision values by 5% for “normal class” and also precision for “attack” class has been increased by 9+%. We have also improved accuracy of classification by 5% over the baseline model.

In any organization, if a connection is flagged as falsely classified as “attack” which, it needs manual review. The increase in the precision of “normal” class would reduce the review time. This is the same case with “attack” class.

The UCB technique is generally used to maximize the model performance by dynamically selecting the best sampling technique. However, in the future we can try different model evaluation metrics like precision, squared loss error on training dataset, the coefficient of determination (r2 value) etc. to compute UCB. This might lead to better UCB results. Yet another experiment can be performed to tune the hyperparameters of UCB and compare cross validation results for the chosen metric.

The training on 5M data points is very slow, and active learning is even slower since it queries interactively for the labels. If we had access to more computing resources and more time to tune and try multiple models, we might have possibly achieved even better results.

CHALLENGES

We have developed the code for active learning from scratch and it is extensible to other projects with minimum possible changes. To run multiple models with huge data and to tune the the hyperparameters using gridsearch is challenging with limited hardware resources. To overcome this, we used the new remote development features in VS Code in tandem with powerful servers on SFU Cloud. This allowed us to run multiple models in a parallel fashion and iterate faster in our search for best hyperparameters for the various models.

REFERENCES

[1] Active learning literature survey Burr Settles, 2009. University of Wisconsin–Madison

[2] Upper-Confidence Bound Policies for Non-Stationary Bandit Problems

[3] A Comprehensive Survey of Machine Learning-Based Network Intrusion Detection

[4] Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system

[5] Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99

[6] The Significant Features of the UNSW-NB15 and the KDD99 Data Sets for Network Intrusion Detection Systems

[7] KDD 99 Dataset Description

[8] PCA filtering and probabilistic SOM for network intrusion detection

[9] A novel hybrid KPCA and SVM with GA model for intrusion detection

[10] Improving the Performance of Multi-class Intrusion Detection Systems using Feature

[11] Network anomaly detection with the restricted Boltzmann machine

[12] Bayesian model averaging of Bayesian network classifiers for intrusion detection

[13] An intrusion detection system using network traffic profiling and online sequent....

[14] A multi-level intrusion detection method for abnormal network behaviors