Intrusion Detection using Machine Learning

Ankit Kumar
Dept. of Mathematics and Computing
Indian Institute of Technology
New Delhi, India
ankit110699@gmail.com

***Abstract*—The rapid growth in the use of computer networks results in the issues of maintaining the network availability, integrity, and confidentiality. This necessitates the network administrators to adopt various types of intrusion detection systems (IDS) that help in monitoring the network traffics for unauthorized and malicious activities. This work uses multiple Machine Learning algorithms to build models for the IEEE BigData 2019 Cup: Suspicious Network Event Recognition dataset. The best model proposed achieves an accuracy of 95% and F1-score of 0.93.**

***Index Terms*—intrusion detection, ids, machine learning**

I. Introduction

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Although intrusion detection systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity.

There are two detection methods of IDS:

1) **Signature-based Method:** Signature-based IDS detects the attacks on the basis of the specific patterns such as number of bytes or number of 1's or number of 0's in the network traffic. It also detects on the basis of the already known malicious instruction sequence that is used by the malware. The detected patterns in the IDS are known as signatures. Signature-based IDS can easily detect the attacks whose pattern (signature) already exists in system but it is quite difficult to detect the new malware attacks as their pattern (signature) is not known.
2) **Anomaly-based Method:** Anomaly-based IDS was introduced to detect the unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in model. Machine learning based method has a better generalized property in comparison to signature-based IDS as these models can be trained according to the applications and hardware configurations.

This work presents Anomaly-based detection method for IDS.

The rest of the paper is organized as follows. Section II discussed related work in this area. Section III describes the dataset used in great detail. Section IV discusses the different models tried, and Section V presents the corresponding results with a brief discussion. Section VI concludes the paper along with possible future work in this direction.

II. Related Work

A review of supervised machine learning techniques is given in [1]. A discussion about the Network Intrusion Detection, techniques and open issues is given in [2]. A detailed survey of the research efforts spared for intrusion detection over last few decades is given in our work [3], with the plenty of works listed in the paper the authors conclude that Hybrid Machine Learning techniques have been used vastly. Authors in [4] proposed the hybrid Audit Data Analysis and Mining, where the anomaly detection is followed by misuse detection. A model based on rule based analysis and statistical model was proposed in [5]. Authors in [6] have examined the feasibility of leaving some of the features of the dataset out, without compromising on the detection rate and had used CFS for feature selection. The detection rate of the reduced dataset was rather improved as CFS selected the best features and eliminated the redundant features. In [7] an approach of misuse detection followed anomaly detection was proposed. To process the large amount of data authors in [8] have proposed anomaly intrusion detection using improved Self Adaptive Bayesian Algorithm.

III. Dataset Description

The dataset used in this work is the IEEE BigData 2019 Cup: Suspicious Network Event Recognition dataset. The following subsections describe the different properties of the dataset, different pre-processing steps, and handling of class imbalance.

A. Description

The dataset has two main files for our use case:

- Cybersecurity Train file: It contains 39427 data points, indexed by alert_ids column (distinct for each data point). Each data point has 63 features, including the alert_ids column and the target column: "notified".
- Localized Alert file: It contains 20 more features, and can be merged with the 1st file. alert_ids column can be used for this task. However, this column is not unique in this case, and the join gives a sequence, which has been described in detail later.

Merging the files gives a new dataframe with 6567968 rows and 83 columns. Before merging, the number of data points with notified = 0 and 1 was 37151 and 2276 respectively, while afterwards it has changed to 3914358 and 2653610 respectively.

1) Missing Values: There are a number of columns with null values. They have been treated in different ways, depending upon their nature:

- 'n1', 'n2', ..., 'n10', 'score': Apart from null values, all the other entries are integers. Thus, the null values have been replaced by 0.
- 'devicetype', 'reportingdevice_code', 'devicevendor_code', 'protocol': All these columns correspond to strings, and the null values have been replaced by "nan" (string).
- 'srcip', 'dstip': These are columns of IP addresses, and neither of them store "0.0.0.0". Thus, all the null values have been replaced by "0.0.0.0".
- 'srcport', 'dstport': These are columns of port numbers, and the null values have been replaced by `max(df_merged[column].unique()) + 1`, since it would have never been used before.

2) Handling of strings: The columns that are of type string are: `['alert_ids', 'categoryname', 'ipcategory_name', 'ipcategory_scope', 'grandparent_category', 'weekday', 'dstipcategory_dominate', 'srcipcategory_dominate', 'alerttype', 'devicetype', 'reportingdevice_code', 'devicevendor_code', 'srcipcategory', 'dstipcategory', 'protocol']`

For all of these, a dictionary has been used to create a mapping between strings and integers, and all the strings have been replaced by corresponding integers.

3) Handling of IP Addresses: Columns `'srcport'` and `'dstport'` store IP addresses. Suppose the address is of the form `''A.B.C.D''`. This can be split to give 4 different columns, with entries being `('A', 'B', 'C', 'D')`. Afterwards, the method described in the previous subsection can be used to convert the entries to integers.

After these three steps, the dataframe contains 6567968 rows and 88 columns.

B. Subset Data

Since $6567968 * 88$ is a huge dataframe (4.4 GB on Google Colab memory), it is important to subset it to be able to build Machine Learning models on this dataset.

1) Sequences: As described very briefly in III-A, merging the two files gives a sequence a points for a given alert id. Although a lot of properties vary among the points, the most intuitive one (that can be used to think of the points as a sequence) is `alerttime`, which is the time at which an alert was sent. Thus, is the data points (for a given alert id) are kept in the sorted order wrt alert time, a sequence of points can be obtained.

Note that the notified label will be the same for all the points in the sequence, since they are derived from the same alert id. During prediction, this property can be very handy. Suppose for a given sequence, x points are classified as 0, while y are classified as 1. Since this should not be the case, appropriate class can be assigned depending upon the values of x and y. Formally, final prediction class = $(y > x)$.

2) Choosing Subset: For any sequence s of length l, define $l_{sub} = min(len(l), 1000)$. Then, l_{sub} random points can be chosen from s.

An advantage of using this subset algorithm instead of just randomly dropping some percentage of the data is that it helps keep a check on the length of sequences while preserving some property from each sequence. A restriction on the length of sequence will go a long way when implementing recurrent networks.

C. Class Imbalance

As mentioned in section III-A, there is some imbalance in the number of data points of the two classes. This subsection addresses these issues. Four datasets have been created using different techniques, as next described. Note that in all of these, the test set is the same, since its not possible to perform such operations on the test set.

1) Dataset 1: The first dataset used in this work is the naive merged dataset as described earlier. It works as a baseline for comparing results for different datasets. The train-val-test is 70-15-15. For all the four datasets mentioned, the number of samples denotes the number of samples in train and val dataset, and not the test dataset. The test dataset is constant, and contains 190005 and 41678 data points for class 0 and 1 respectively.

- Number of samples of class 0: 1103439
- Number of samples of class 1: 209429

2) Dataset 2: Here, the Synthetic Minority Oversampling Technique (SMOTE) [9] has been used to increase the samples of the minorty class.

- Number of samples of class 0: 1103439
- Number of samples of class 1: 1103439

3) Dataset 3: Here, Edited Nearest Neighbours (ENN) [10], an undersampling approach has been used. It applies a nearest-neighbors algorithm and "edit" the dataset by removing samples which do not agree "enough" with their neighborhood. It reduces the size of the original dataset.

- Number of samples of class 0: 1046695
- Number of samples of class 1: 209429

4) Dataset 4: Finally, a combination of both SMOTE and ENN is used [11]. SMOTE can generate noisy samples by interpolating new points between marginal outliers and inliers. This issue can be solved by cleaning the space resulting from over-sampling – exactly where ENN comes into play.

- Number of samples of class 0: 1086962
- Number of samples of class 1: 1034287

D. Curse of Dimensionality

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. One of those is the distance between points. Too many dimensions cause every observation in the dataset to appear equidistant from all the others. And because algorithms like clustering uses a distance measure such as Euclidean distance to quantify the similarity between observations, this is a big problem. If the distances are all approximately equal, then all the observations appear equally alike (as well as equally different), and no meaningful clusters can be formed.

Thus, for each dataset, the dataset is reduced to `nComp` dimensions, where `nComp` is an integer in the range $[4, 40]$ (described in more details in Section IV). Rest of the architectures remains the same, and performance is measured – whether PCA increases or decreases model accuracy.

IV. Models

This section describes the different models that have been used in this work.

A. Common Approach

For all the models described in the subsequent subsections, the approach described here is common. Each model is run on all 4 datasets described in section III-C. For each dataset, there are 3 further possible modes to run:

1) With normalization (z score normalization)
2) Without normalization
3) With PCA
   For PCA, there are two ways to run:
   - Number of components = 20 (used when training model takes too long)
   - Number of components from 4 to 40 with step size of 6 (used when training model is relatively quick)

Thus, each model is run for each dataset in 3 different modes. Given a model, for each dataset, the best mode can be chosen (which gives the best results). After that, the best dataset can be chosen, and different models can be compared based on the best score they are able to obtain across datasets and modes.

B. Naive Bayes

In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. In this work, variance smoothing has been varied to find the best suitable hyper-parameter.

C. K-Means

K-Means clustering is a method of vector quantization,, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. In this work, number of clusters have been varied to find the best suitable hyper-parameter.

In the implementation of K-Means, Batch K-Means has been used with batch size = 1000 because of memory issues in absence of batch processing.

D. Logistic Regression

In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This work assumes 'l2' penalty (the norm used in the penalization).

E. Artificial Neural Network

An artificial neural network (ANN) is the component of artificial intelligence that is meant to simulate the functioning of a human brain. In theory, ANNs are capable of generalizing any given function (assuming ANN can have as many layers as needed for this task). In this work, the number of hidden layers, optimizer used and number of epochs have been varied to find the best suitable hyper-parameters.

F. Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. In this work, criterion (the function to measure the quality of a split) and splitter (the strategy used to choose the split at each node) have been varied to find the best suitable hyper-parameters.

G. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees. In this work, criterion (the function to measure the quality of a split) and n_estimators (the number of trees in the forest) have been varied to find the best suitable hyper-parameters.

H. XGBoost

XGBoost [12] stands for "Extreme Gradient Boosting", where the term "Gradient Boosting". It is is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In this work, maximum depth of the tree has been varied to find the best suitable hyper-parameter.

I. AdaBoost

AdaBoost [13], short for Adaptive Boosting, is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps to combine multiple "weak classifier" into a single "strong classifier" (the weak learners in AdaBoost are decision trees with a single split, called decision stumps). In this work, learning rate (weight applied to each classifier at each boosting iteration – a higher learning rate increases the contribution of each classifier) and n_estimators (the maximum number of estimators at which boosting is terminated) have been varied to find the best suitable hyper-parameters.

J. LightGBM

LGBM [14], short for Light Gradient Boosted Machine, is a library developed at Microsoft that provides an efficient implementation of the gradient boosting algorithm. LightGBM does not grow a tree level-wise (row by row) as most other implementations do. Instead it grows trees leaf-wise. It chooses the leaf it believes will yield the largest decrease in loss. The primary benefit of the LightGBM is such changes to the training algorithm that make the process dramatically faster, and in many cases, results in a more effective model. In this work, learning rate has been varied to find the best suitable hyper-parameter.

K. CatBoost

CatBoost [15] is a third-party library developed at Yandex that provides an efficient implementation of the gradient boosting algorithm. The primary benefits of the CatBoost are that it has ordered Boosting to overcome over fitting, it allows to use non-numeric factors (instead of having to pre-process to turn it to numbers) and the use of symmetric trees for faster execution. In this work, n_estimators (the maximum number of trees that can be built) have been varied to find the best suitable hyper-parameter.

V. Evaluation

This section presents the evaluation metrics used in this work, and the results corresponding to the different models discussed in Section IV.

A. Evaluation Criteria

For each model, three metrics have been reported:

1) Accuracy
2) F1-Score
3) AUC-ROC

For each model, various hyper-parameters have been tried. In order to choose the best hyper-parameter(s), the same order of evaluation metrics as above have been used. That is, first accuracy is checked. If its same, F1-Score is checked, and even if thats same, AUC-ROC is checked.

For each model, only the best results over all the datasets and all the different modes are given. In case of tie, all the results are reported.

B. Naive Bayes

- Dataset: $1, 3$
- Mode: 3
- Accuracy: 0.94
- F1-score: 0.92
- AUC-ROC: 0.6532

C. K-Means

- Dataset: $1, 3$
- Mode: $1, 2, 3$
- Accuracy: 0.94
- F1-score: 0.91
- AUC-ROC: 0.5917

D. Logistic Regression

- Dataset: 3
- Mode: 2
- Accuracy: 0.94
- F1-score: 0.91
- AUC-ROC: 0.5850

E. Artificial Neural Network

- Dataset: $1, 3$
- Mode: 2
- Accuracy: 0.94
- F1-score: 0.92
- AUC-ROC: 0.6702

F. Decision Tree

- Dataset: 1
- Mode: 2
- Accuracy: 0.93
- F1-score: 0.93
- AUC-ROC: 0.6587

G. Random Forest

- Dataset: 1
- Mode: 2
- Accuracy: 0.95
- F1-score: 0.93
- AUC-ROC: 0.7709

H. XGBoost

- Dataset: 3
- Mode: 2
- Accuracy: 0.94
- F1-score: 0.93
- AUC-ROC: 0.7341

I. AdaBoost

- Dataset: 3
- Mode: 1
- Accuracy: 0.94
- F1-score: 0.92
- AUC-ROC: 0.6960

J. LightGBM

- Dataset: 1
- Mode: 2
- Accuracy: 0.95
- F1-score: 0.93
- AUC-ROC: 0.7614

K. CatBoost

- Dataset: 3
- Mode: 2
- Accuracy: 0.94
- F1-score: 0.93
- AUC-ROC: 0.7350

L. Key Points

- PCA was useful only for the case of Naive Bayes, when the best models trained after dimensionality reduction were better than the rest.
- Models trained on dataset 2 and 4 were never the best, hinting that using SMOTE may lead to nosiy data.
- Models trained on dataset 3 were the best 7 times, while the ones trained on dataset 1 were the best 6 times. Thus, undersampling using ENN (to filter the "noisy" data points of the class with higher number of points) is better than using the given dataset as is.
- While both Random Forest and LGBM have the same accuracy and F1-score (0.95 and 0.93 respectively), AUC for Random Forest (0.7709) is better than the one for LGBM (0.7614). Thus, Random Forest is the best model to use out of the models discussed in this work.
- Interestingly, both Random Forest and LGBM perform best on dataset 1 (dataset with class imbalance in training and validation).
- Most of the models perform the best in the mode = 2 setting, in which the dataset is used without any normalization.

M. Proposed Pipeline

In view of the results described above, the following pipeline can be used to train a model and predict on an unknown dataset:

1) Merge and clean the dataset as described in Section III-A.
2) (Optional depending upon computational resources) Subset the data according to the strategy described in Section III-B.
3) Perform an 80-20 split for the train and validation datasets.
4) Choose Random Forest to be the prediction model.
5) For hyperparameter tuning, train using the train dataset, and choose the best hyperparameter (to be used for the test dataset) using the validation dataset.
6) Combine both the train and validation datasets. Train using the (best) hyperparameter obtained in previous step, and predict for the test set.

VI. Conclusion and Future Work

This work proposed different models for the IEEE BigData 2019 Cup: Suspicious Network Event Recognition dataset. Out of all the models proposed, Random Forest performs the best on all three all three metrics: accuracy (0.95), F1-score (0.93) and AUC-ROC (0.7709). Thus, atleast out of the models discussed in this work, Random Forest should be the go to model for Intrusion Detection using Machine Learning.

Further, models trained after filtering the dataset using Edited Nearest Neighbours (ENN) outperform the rest (Section V-L), showing that it is better to filter the data (assuming there are "enough" points) than producing noisy data points.

In continuation to the previous point, using SMOTE or a combination of SMOTE or ENN results in noisy samples. Thus, this may not be the best method to handle class imbalance if more samples are desired, and neural generative frameworks may be the way to go. The space of neural generative frameworks is dominated by Generative Adversarial Networks (GANs) [16] and its variants. Although GANs can generate realistic novel examples, in a vanilla GAN, it is not possible to control the types or class of images that are generated. The conditional generative adversarial network (cGAN) [17] is a modified framework of GANs suitable for conditional generation of data by the generator network. Data generation can be conditioned using the class label, in a supervised manner, allowing the targeted generation of data of a given type. This can be used to generate more samples of the class with the lesser number of samples instead of SMOTE (or related variants).

In this work, ensemble models haven't been used, mainly because of the lack of computational resources. Since lot of models perform really well (both LGBM and Random Forest have the highest accuracy and F1-score), an ensemble model can be tried so as to try to improve the predictions.

Finally, Section III-B discusses the sequences that are present in the data. We are given a sequence of vectors, and we need to give a boolean prediction. Recurrent Neural Networks (vanilla RNN and other variants like LSTMs) are extremely suitable models for this purpose, and can be experimented with in the future.

References

[1] S. B. Kotsiantis, I. Zaharakis and P. Pintelas, "Supervised machine learning: A review of classification techniques", 2007.

[2] C. A. Catania and C. G. Garino, "Automatic network intrusion detection: Current techniques and open issues", 2012.

[3] Y. Hamid, M. Sugumaran and V. Balasaraswathi, "IDS Using Machine Learning - Current State of Art and Future Directions", 2016.

[4] D. Barbara, J. Couto, S. Jajodia, L. Popyack and N. Wu, "ADAM: Detecting intrusions by data mining", 2001.

[5] D. Anderson, T. Frivold and A. Valdes, “Next-generation intrusion detection expert system (NIDES): A summary”, 1995.
[6] M. A. Hall, “Correlation-based feature selection for machine learning”, 1999.
[7] J. Zhang and M. Zulkernine, “A hybrid network intrusion detection technique using random forests”, 2006.
[8] D. M. Farid and M. Z. Rahman, “Anomaly network intrusion detection based on improved self adaptive bayesian algorithm”, 2010.
[9] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall and W Philip Kegelmeyer, “Smote: synthetic minority over-sampling technique”, 2002
[10] Dennis L Wilson, “Asymptotic properties of nearest neighbor rules using edited data”, 1972.
[11] Gustavo EAPA Batista, Ana LC Bazzan and Maria Carolina Monard, “Balancing training data for automated annotation of keywords: a case study”, 2003.
[12] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System”, 2016.
[13] Y. Freund and R. E. Schapire, “A Short Introduction to Boosting”, 1999.
[14] Guolin Ke1, Qi Meng2, Thomas Finley3, Taifeng Wang1, Wei Chen1, Weidong Ma1, Qiwei Ye1 and Tie-Yan Liu1, ”LightGBM: A Highly Efficient Gradient Boosting Decision Tree”, 2017.
[15] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush and Andrey Gulin, ”CatBoost: unbiased boosting with categorical features”, 2017.
[16] I. Goodfellow, J. P. Abadie, M. Mirza, B. Xu, D. W. Farley, S. Ozair, A. Courville and Y. Bengio, “Generative adversarial nets”, 2014.
[17] M. Mirza and S. Osindero. “Conditional generative adversarial nets”, 2014.