Network Based Intrusion Detection

Rahul Yadav
Department of Computer Science and Engineering University at Buffalo, Buffalo, NY 14260
rahulyad@buffalo.edu

1. Intrusion Detection

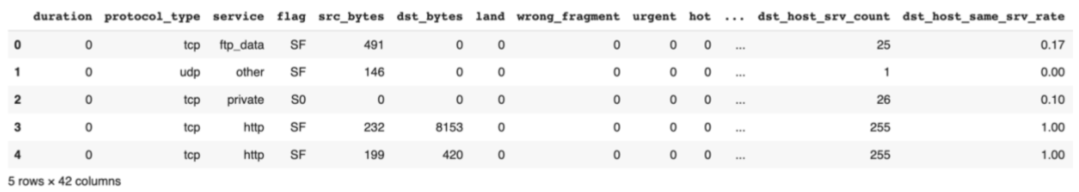
Intrusion detection is the process of monitoring computer systems, networks, and other digital assets to identify unauthorized access or activity. The goal of intrusion detection is to detect and respond to security threats as quickly as possible, to minimize the potential damage and protect valuable data. There are two main types of intrusion detection:

1. Host-based intrusion detection (HIDS): This type of intrusion detection focuses on monitoring individual devices, such as servers or workstations, to identify suspicious activity that could indicate an intrusion. HIDS systems typically work by analyzing system logs, file integrity, and other indicators of unusual activity.
2. Network-based intrusion detection (NIDS): This type of intrusion detection monitors network traffic to identify suspicious patterns or anomalies that could indicate an intrusion. NIDS systems typically work by analyzing network packets and other network traffic data.

In our experimentation we talk about Network-based intrusion detection (NIDS) on KDD dataset. The KDD Cup 1999 dataset is a popular benchmark dataset for Network-based Intrusion Detection Systems (NIDS). There are various machine learning (like SVMs, Random Forest) and deep learning models (like RNNs and ANNs) that can be used to classify network traffic in this dataset.
Overall, intrusion detection plays a critical role in maintaining the security of digital assets and protecting against cyber-attacks. By quickly identifying and responding to potential threats, intrusion detection can help organizations minimize the risk of data loss, downtime, and other negative consequences of security breaches.

2. Dataset

The KDD Cup 1999 dataset consists of a set of network traffic features that were collected from a simulated environment, designed to mimic a typical network infrastructure. The simulated environment included several hosts connected to a hub, which in turn was connected to an Internet Service Provider (ISP). The hosts represented different services, such as web servers, mail servers, and FTP servers.
The network traffic was generated using several attack scenarios, designed to simulate common types of network attacks, such as denial-of-service (DoS), probing, and user-to-root (U2R) attacks. The attacks were performed using several tools, such as smurf, neptune, and warez. The dataset includes both normal and attack traffic, with a total of 1.25 million records.
Each record in the dataset represents a network connection and includes 41 different features. The features include basic network statistics, such as the number of bytes and packets sent and received, as well as more advanced features, such as the type of protocol used, the service being accessed, and the duration of the connection. The features were selected based on their relevance to intrusion detection, as well as their ability to distinguish between normal and attack traffic.



	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00

5 rows × 42 columns

Figure 1: Dataset View

2.1. Data Preprocessing:

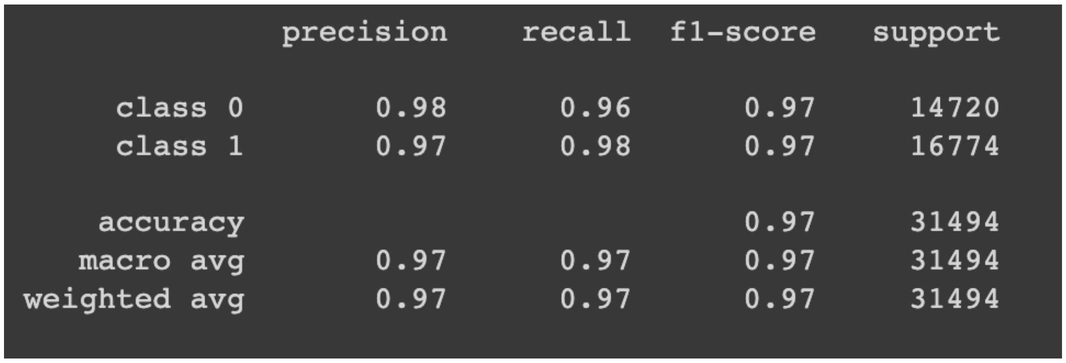
- Multiple class to binary classes:
  Types of attacks (Intrusion): neptune, satan, ipsweep, portsweep, smurf, nmap, back, teardrop, warezclient, pod, guess_passwd, buffer_overflow, warezmaster, land, imap, rootkit, loadmodule, ftp_write, multihop, phf, perl, spy.
  But to simplify this problem and balance the classes we modified the labels into only two class normal and abnormal (where normal accounts for 67343 records and abnormal accounts for 58630 records.
- Checked for missing values in the dataset, didn't found any missing values.
- Numeric feature normalization of the dataset using Min-Max Scalar
- Categorical feature handling using pandas get_dummies function, which convert categorical variables into dummy or indicator variables. A dummy or indicator variable can have a value of 0 or 1.
- Converting labels to binary values using label_encoder.

After performing the above preprocessing steps on the KDD training dataset, it is ready for modeling and training a classifier for the binary classification problem.

3. Experiment

In this experiment we build a binary classifier to accurately that an input record is normal or abnormal(intrusion) with given attributes has diabetics or not? As we are given with labeled binary_label (target variable), it is supervised classification problem.
We trained two types of classifiers for this problem, one was a Linear Support Vector Machines(SVM) and other was a Multi-Layer Perceptron Classifier. We have compared the results of both of these classifier below:

3.1 Results from Linear SVM classifier:



```
              precision    recall  f1-score   support

     class 0       0.98      0.96      0.97     14720
     class 1       0.97      0.98      0.97     16774

    accuracy                           0.97     31494
   macro avg       0.97      0.97      0.97     31494
weighted avg       0.97      0.97      0.97     31494
```

Figure 2: F1 Score of Linear SVM Classifier

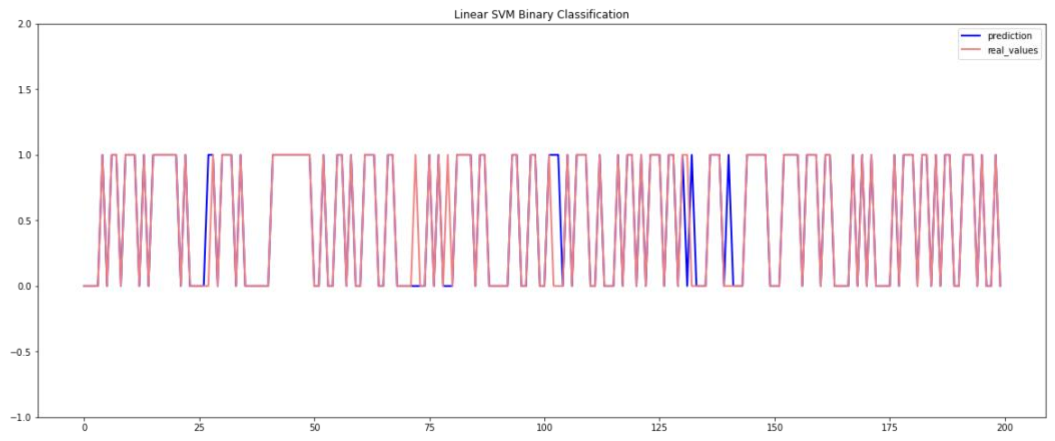


Figure 3: Prediction on test data by Linear SVM classifier

3.2 Results from Multi-Layer Perceptron classifier:

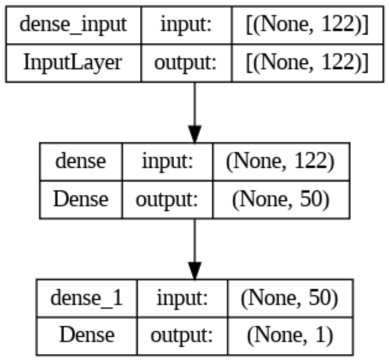

dense_input | input: | [(None, 122)]
InputLayer | output: | [(None, 122)]
dense | input: | (None, 122)
Dense | output: | (None, 50)
dense_1 | input: | (None, 50)
Dense | output: | (None, 1)

Figure 4: Layer Architecture of MLP Classifier

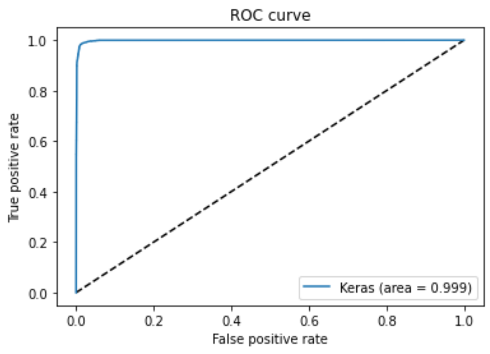


Figure 5: ROC MLP classifier

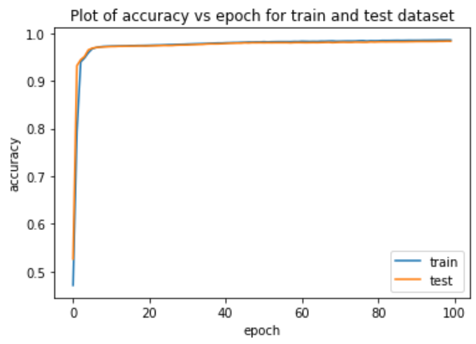


Figure 6: Accuracy vs Epoch of MLP Classifier

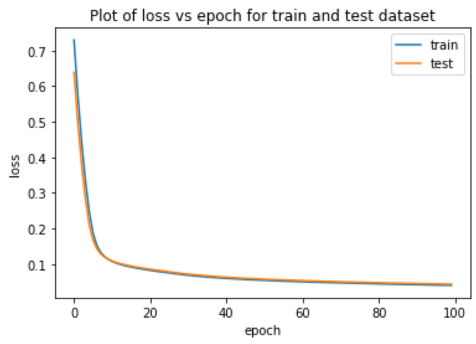


Figure 7: Loss vs Epoch of MLP Classifier

3.3 Observation:

Although the KDD Cup 1999 dataset can be used to train a highly accurate classifier for detecting network-based intrusions, it is important to note that this dataset is now outdated and does not reflect the current state of network security. Unfortunately, there are not many real-world intrusion detection datasets that have enough data points, which makes it challenging to train effective intrusion detection systems. Therefore, it is necessary to come up with alternative approaches to train intrusion detection systems that consider the scarcity of available data.
So, we experimented to use the CTGAN to generate synthetic data for intrusion detection and supplement the limited data with the synthetically generated data.

4. Synthetic Data Generation with CTGAN

CTGAN is a state-of-the-art generative adversarial network (GAN) that can be used to generate synthetic data that mimics the distribution of real-world data. This approach can be used to generate synthetic data for intrusion detection when the available data is limited.
In our experiment we used a small subset of the KDD dataset (3000 records) and generated 1000 synthetic records using trained CTGAN model.
In the initial look the data looked very similar to the original the dataset but when we trained a linear SVM classifier and used it for prediction, we observed very poor performance on the unseen data records. Please check the figure below (figure 8) to observe how bad was the classifier's performance on the test set.

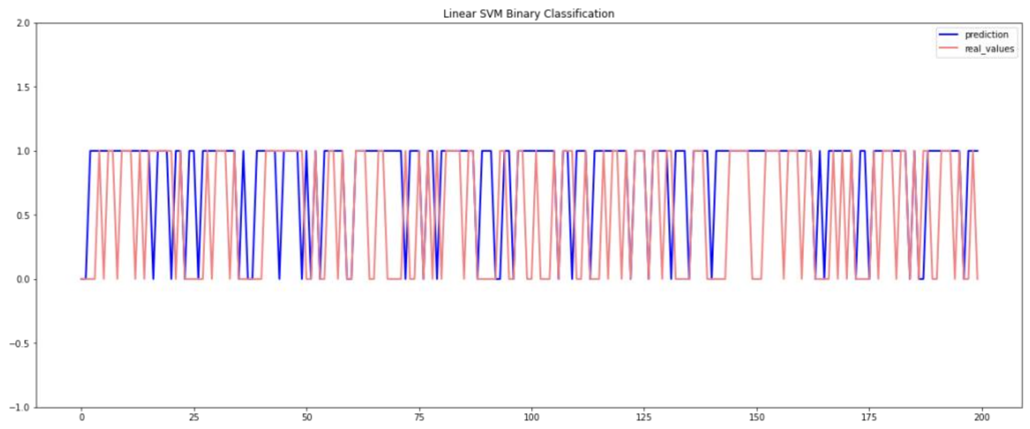
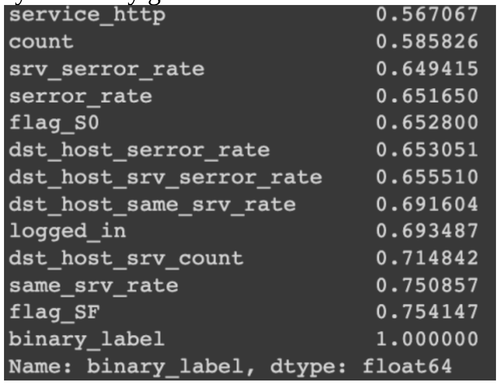
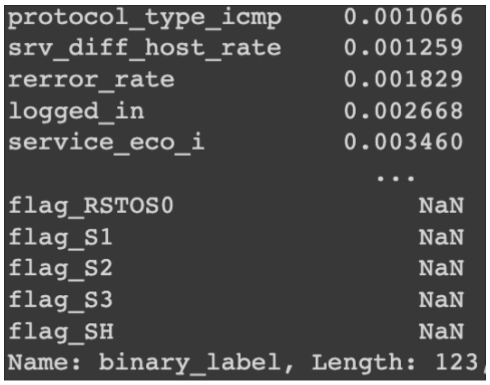


Figure 8: Prediction of Linear SVM Classifier trained on synthetically generated data.

When I further analyzed the reason of such poor performance, I identified that the synthetic data generated by CTGAN does not consider the feature correlation within different features and that might lead to in-accurate data representation. Please check the figures below to observe the differences between the feature correlation in the original dataset and the feature correlation in the synthetically generated dataset.


```
service_http               0.567067
count                      0.585826
srv_serror_rate            0.649415
serror_rate                0.651650
flag_S0                    0.652800
dst_host_serror_rate       0.653051
dst_host_srv_serror_rate   0.655510
dst_host_same_srv_rate     0.691604
logged_in                  0.693487
dst_host_srv_count         0.714842
same_srv_rate              0.750857
flag_SF                    0.754147
binary_label               1.000000
Name: binary_label, dtype: float64
```
Figure 9: Top 10 most correlated features with the target label in the original dataset


```
protocol_type_icmp     0.001066
srv_diff_host_rate     0.001259
rerror_rate            0.001829
logged_in              0.002668
service_eco_i          0.003460
                          ...
flag_RSTOS0                 NaN
flag_S1                     NaN
flag_S2                     NaN
flag_S3                     NaN
flag_SH                     NaN
Name: binary_label, Length: 123
```
Figure 10: Correlation of all features with the target label in the synthetic dataset

We further observed the differences using a heatmap of correlations:

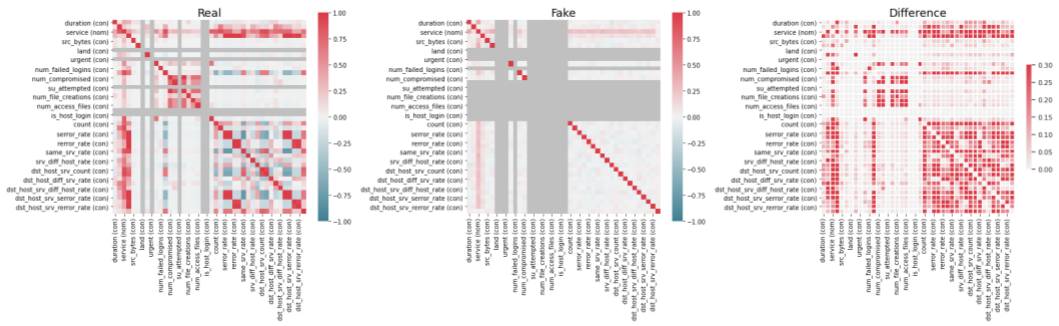


Figure 11: Heatmap of correlation in Original and Synthetic (Fake) Dataset, the difference of these two correlations

We visualize the difference better we can reduce the high dimensional data and just compare the 2 principal components using PCA.

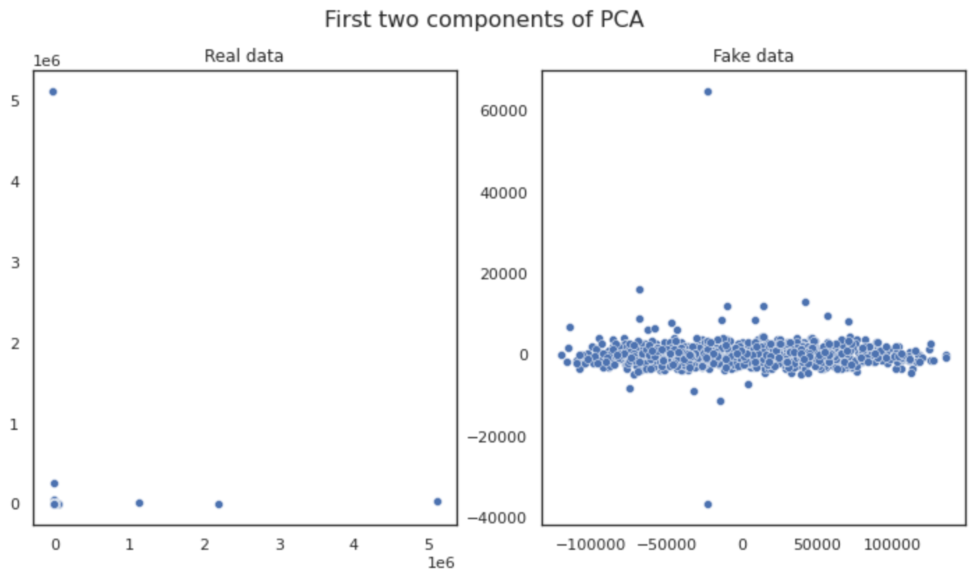


Figure 12: Two Principal Components of the Original Dataset vs the Synthetic Dataset

5. Conclusion:

Based on the experiment, it can be concluded that synthetic data generation is not a direct alternative for supplementing limited data in intrusion detection datasets. This is because synthetic data generation techniques, such as CTGAN, may not maintain the feature correlations present in the original dataset, resulting in each feature being generated as an independent attribute. However, in most real-world datasets, features are correlated, and their relationships must be preserved to accurately represent the data. As a result, it is necessary to come up with better alternatives to address the problem of limited data in intrusion detection datasets.

One such alternative is to treat the intrusion detection problem as anomaly detection problem (one-class classification) and model the problem using autoencoders.

Autoencoders are a type of unsupervised learning technique that can be used to learn the underlying patterns and structure of the data without the need for labeled data. In the context of intrusion detection, autoencoders can be trained on a limited dataset of normal network traffic, and then used to detect anomalous or malicious network traffic that deviates from the learned normal patterns.

The basic idea is to train the autoencoder on normal network traffic, which is assumed to be representative of most of the traffic. During training, the autoencoder learns to encode the input data into a low-dimensional representation (latent space) and decode it back to the original input. The encoder and decoder components of the autoencoder can then be used to identify anomalous or malicious traffic by comparing the reconstruction error of the input with the original input.

However, it is important to note that autoencoders have some limitations. For instance, autoencoders may not perform well when the input data has high levels of noise or when there are multiple types of anomalies present in the data. Therefore, after the unsupervised learning we can test the performance of our autoencoder on the labelled dataset available to us, this dataset will work as the gold standard for our model.

6. References:

[1] “NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB.” *NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB*, www.unb.ca/cic/datasets/nsl.html.

[2] abhinav-bhardwaj. “GitHub - Abhinav-bhardwaj/Network-Intrusion-Detection-Using-Machine-Learning: A Novel Statistical Analysis and Autoencoder Driven Intelligent Intrusion Detection Approach.” *GitHub*, 12 Oct. 2021

[3] Asharf, J.; Moustafa, N.; Khurshid, H.; Debie, E.; Haider, W.; Wahab, A. A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions. *Electronics* **2020**, *9*, 1177. https://doi.org/10.3390/electronics9071177

[4] Ferriyan, A.; Thamrin, A.H.; Takeda, K.; Murai, J. Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic. *Appl. Sci.* **2021**, *11*, 7868. https://doi.org/10.3390/app11177868

[5] Chalapathy, Raghavendra, and Sanjay Chawla. “Deep Learning for Anomaly Detection: A Survey.” arXiv.org, 10 Jan. 2019, https://doi.org/10.48550/arXiv.1901.03407.

[6] “Intro to Autoencoders | TensorFlow Core.” *TensorFlow*, www.tensorflow.org/tutorials/generative/autoencoder.