School of
Computing and
Information Systems

ISSS610 Applied Machine Learning

DDoS Attack Detection Using Machine Learning

By Group 3:
Aishwarya KRISHNA PRASAD, HAI Dan, LAN Chengcheng, LU Di, Nicole TAY Zi Min

Table of Contents

1.Introduction

A Distributed Denial of Service (DDoS) attack is a menace to network security that aims at exhausting the target networks with malicious traffic. It is one of the most common and dangerous types of cybersecurity attacks. DDoS attacks have been on the rise and are becoming more complex. Many statistical methods have been designed for DDoS attack detection, but there is still a business need to design one that can operate in real-time and done at a low cost. As such, we aim to come up with models to not just to detect the various DDoS attack types, and to instead be flexible and detect any anomalies in network traffic that may spell new types of DDoS attacks that may surface in the future.

2.Dataset

We utilize a dataset called CICDDoS2019 (quote source), which contains a comprehensive variety of DDoS attacks and information extracted. The training data contains 11 different kinds of DDoS attacks, recorded in 11 separate files. The testing data contains 7 DDoS attacks, recorded in 7 separate files, totaling up to 18 separate files. One type of DDoS attack, PortScan, is only present in the testing dataset unlike the rest which were present in both. This helped to ensure that any models trained do not overfit the training data.

CICDDoS2019 has 87 flow-based features and 1 target variable – 'Benign' or 'Attack', giving us a binary classification problem. This dataset was found to be imbalanced across both classes and has over 50 million rows.

3.Data Pre-Processing

First, we opted to drop the 7 static features present: *'Flow ID', 'Source IP', 'Source Port', 'Destination IP', 'Destination Port', 'Protocol'* and *'Timestamp'*, as we are unable to differentiate malicious and normal users based on these features (Elsayed, Le-Khac, Dev, & Jurcut, 2020). We then decided to drop 13 constant features. They ~~were either duplicates of other columns or~~ had a constant value for all rows. Such constant features are not useful for differentiating malicious and normal network traffic, generates noise in the data and may decrease the performance of the machine learning model if included. Subsequently, we dropped 6 duplicate features. These features have mostly identical values to other features, so we kept the first instance of that feature, then dropped the duplicates. They are *'RST Flag Count', 'Fwd Header Length', 'Subflow Fwd Packets', 'Subflow Fwd Bytes', 'Subflow Bwd Packets'*, and *'Subflow Bwd Bytes'*. This leaves us with 60 features and 1 target variable.

We removed 4,603 rows with missing values across all 18 datasets. The datasets were highly imbalanced as more than 99.9% of total records were labelled as 'Attack'. To resolve this issue as much as possible, in each dataset, we reserved only 20,000 'Attack' instances but kept all 'Benign' instances. After combining all training and testing datasets, proportion of 'Benign' instances increased sizably (Table 1). The overall datasets size also dropped significantly (Table 2) which resulted in much lesser strain on our limited computational resources.

	Training Datasets		Testing Datasets	
Network Traffic Type	Before	After	Before	After
Attack	99.9%	86.4%	99.9%	70.4%
Benign	0.1%	13.6%	0.1%	29.6%

Table 1: Class Proportion in Training and Testing Datasets

Network Traffic Type	Training Datasets Size	Testing Datasets Size
Attack	215,192	133,868
Benign	34,036	56,308

Table 2: Size of Final, Combined Datasets

Logistic Regression

In logistic regression, we used GridSearchCV() to find the optimal value of the parameters, they are 15 for C, fit_intercept as true and penalty as l2. The model performance on validation and testing datasets is shown below.

Evaluation Metrics	Validation Datasets	Test Datasets
Precision	0.9494	**0.8512**
Recall	0.9980	0.9901
F1 Score	0.9731	0.9154
AUC	0.9471	0.9120

Table 3: Logistic Regression Performances

Confusion Matrix (Test Dataset)		
	Predicted Benign	Predicted Attack
Actual Benign	33130	**23176**
Actual Attack	1320	132548

Table 4: Confusion Matrix of Logistic Regression

The precision in test datasets of logistic regression is only 0.8512 and have 23176 rows data are Benign but predicted as Attack. Although the model has a greater performance on recall in test datasets which was 0.9901. But in cybersecurity, we need the model to detect as many DDoS attacks as possible, while the precision is good enough to ensure the model is highly efficient. The logistic regression will be our baseline and other models in our team need performance better than this one.

Ensemble Models

We have done the ensemble models for Randomforest, Adaboost, Gradientboosting, and XGboost. Firstly, we use Randomized Search to tune the hyperparameters which are shown in the table.

Random Forest Classifier	n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	criterion	bootstrap
Best Parameters	200	8	1	sqrt	None	entropy	True

AdaBoost Classifier	n_estimators	learning_rate	algorithm
Best Parameters	50	1	SAMME.R

Gradient Boosting Classifier	n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	learning_rate
Best Parameters	150	2	1	auto	5	0.1

XGBoost Classifier	min_child_weight	gamma	max_depth	learning_rate	colsample_bytree
Best Parameters	1	8	6	0.3	0.5

Table 5: Hyperparameter Tuning

There is no big difference, and scores are high for these ensemble models.

Evaluation Metrics	Validation Datasets	Test Datasets
Precision	0.99995	0.9964
Recall	0.9998	0.9988
F1 Score	0.9999	0.9976
AUC	0.9997	0.9951

Table 6: Random Forest Classifier Evaluation Metrics

Evaluation Metrics	Validation Datasets	Test Datasets
Precision	0.9995	0.9839
Recall	0.9996	0.9996
F1 Score	0.9996	0.9917
AUC	0.9983	0.9804

Table 7: AdaBoost Classifier Evaluation Metrics

Evaluation Metrics	Validation Datasets	Test Datasets
Precision	0.9999	0.9914
Recall	0.9997	0.9999
F1 Score	0.9998	0.9957
AUC	0.9996	0.9897

Table 8: Gradient Boosting Classifier Evaluation Metrics

Evaluation Metrics	Validation Datasets	Test Datasets
Precision	0.99995	0.9967
Recall	0.9998	0.9999
F1 Score	0.9999	0.9983
AUC	0.9997	0.9960

Table 9: XGBoost Classifier Evaluation Metrics

Then we look at feature importance. Overall, different models appear to favour different features.

Top3	Random Forest	AdaBoost	Gradient Boosting	XGBoost
1	Inbound	Init_Win_bytes_backward	Inbound	Inbound
2	Bwd Packets/s	Init_Win_bytes_foward	URG Flag Count	URG Flag Count
3	Total Backward Packets	Flow IAT Std	Init_Win_bytes_forward	CWE Flag Count

Table 10: Feature Importance

It is important to note that most important features are related to derived statistical features of packets sent or abnormal flags. For example, the top 1 feature is Inbound for three models. Inbound means Inbound traffic originates from outside the network -- the internet. Information coming into a network -- computer network. These features make business sense because attack is like to happen for Inbound and would be closely monitored.

ANN with Feed Forward Process

After the random forest model and adaboost model, our group decide to create artificial neural network with feed forward process to detect the DoS attack and DDoS attack. The reason we want to choose deep learning model is because deep learn model have achieved high significant due to their efficient performance, especially CNN model in image processing and RNN in sequential data. Since our final target to create a model with well performance classifier, ANN with feed forward structure is the best fit the scenario.

After doing the data preprocessing, the first step is using the sklearn preprocessing package to do a normalization. We build a two layers ANN model with the input layer with dimension of 60 feature and we add 32 neurons in this layer. The first layer using the activation function with relu, because relu is one of the most frequent use functions in the neural network. And we choose learning rate equal 0.01 and batch size equal to 256. The reason we use this combination is because it gives me the lowest loss on test and validation date. Also, we choose epoch equal to 50 to gain the train loss and validation from history.

```
def create_model(my_learning_rate):
  # initial ANN
  model = Sequential()

# layers
  model.add(Dense(units = 32,  activation = 'relu', input_dim = 60))

  model.add(Dense(units = 1, activation = 'sigmoid'))

  opt = Adam(learning_rate= my_learning_rate)
  model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

  return model
```

Figure i

After build the baseline model, we get the accuracy as 99.7% and loss score equal to 0.012. And we plot out the graph on training loss and validation loss. There don't have obvious overfitting. We find out that there are 47 case of FN and 492 case of FP, we want to further reduce the FPR(False positive rate) and even to increase the precision score.

Confusion Matrix (Test Dataset)		
	Predict Benign	Predict Attack
Actual Benign	55814(TN)	492(FP)
Actual Attack	47(FN)	133821(TP)

Figure ii

Next steps are following by two deeper model bases on the baseline model. One deeper model is total with 5 layers and another deeper model is total with 5 layer and 2 layer of dropout regularization. From the confusion matrix, the deeper model further reduces the FP number to 231 and the accuracy rate increase to 99.9%. Even though there don't show the obvious overfitting in the train loss in the figure(iii), we want to see if there have further improvement. In the deeper model with dropout regularization, we add to dropout layer with 10% and 20%. In the end, we choose deeper model with dropout

regularization as our representation model in ANN. It shows the loss on test data is 0.031 and accuracy rate as 99.9%. Total FP was reduced to 196, lower than the earlier deeper model.

Confusion Matrix (Test Dataset)		
	Predict benign	Predict Attack
Actual Benign	56075	231
Actual Attack	46	133822

Confusion Matrix (Test Dataset)		
	Predict Benign	Predict Attack
Actual Benign	56110	196
Actual Attack	46	133822

Figure iii

Semi Supervised Learning using Auto Encoders and SVM

While implementing the models above, we realized that we got a dataset that was labelled with millions of records. But in a real world- it's very difficult to label the datasets by ourselves. In most cases the datasets used for research (like ours) have been curated by scientists who work on it for month or even years to label them manually. Not to forget the DDoS attacks can 'mutate' in the future. We realized that our models should not be limited to the DDoS attack patterns that are documented in this dataset alone. Also, our dataset contains majority of DDoS attack transactions compared to normal transactions, but in reality, the number of DDoS attacks are way lesser than normal transactions that occur in internet. Also, hackers find a way to implement different variants of DDoS attacks – and we realized a need to implement a generic model that would differentiate between DDoS attacks and normal transactions without having the need to label too many datasets (use the existing labelled dataset), work around the class imbalance issues and detect new DDoS attack patterns that are not present in our dataset. We used autoencoders for the unsupervised part of our model and SVM kernels for the supervised part in our experiments. (Gogna & Majumdar, (2016, October)).

As part of data pre-processing, the input values were standardized using Minmax scalar from sklearn, the rows with infinity and na values were removed in order to standardize the values. We ran around 4 experiments, with different variations autoencoders architectures for training our model. Common to all the methods above, we split our dataset to training and testing parts. From the training set, 15,000 normal transaction records were used to train our autoencoder models. The intuition behind is approach is, if we train our autoencoders with normal transactions, the encoder would differentiate the representation if it encounters any transaction that is not normal (which would cover all kinds of DDoS attack transactions).The 4 models had the following architectures Dense nodes (200,100,50,30,10,30,50,100,200) , Dense (200,100,50, 30,50,100,200) , Dense (200,100,75,50,30,10,30,50,75,100,200) for the models 3 and 4. Model 1 was trained for 10 epochs – we found that even with 10 epochs, the autoencoder had managed to learn some representations. The model 2 and model 3 were trained for 2000 epochs, it was observed that model three performed better than model 2. The gradual increase and decrease in the number of dense nodes in each layer for model 3, certainly helped the model learn better. But it was noticed that the validation and training loss continued to drop even for 2000 epochs, hence the model 4 was trained for 4000 epochs with the same architecture as model 3 with early stop parameters as monitor='val_loss', min_delta=0.00001, patience=50, verbose=1, mode='min', restore_best_weights=True. We observed that model 4 performed the best and the validation and training errors were still dropping steadily for over 4000 epochs. 19036 Benign transactions and rest of the data (DDoS attack from train set) were fitted separately to obtain the latent representation of our DDoS attack and normal transactions. We built a TNSE plot to observe if the representations are linearly separated (Fig 1.1) we observe that the representations are not linearly separated. We then employ the SVM kernels with C=20, to learn from the representation obtained from model 4. The test data set is used as a ground truth, to validate the representation learnt from the SVM kernel and this approach led to the results stated in Table 11.

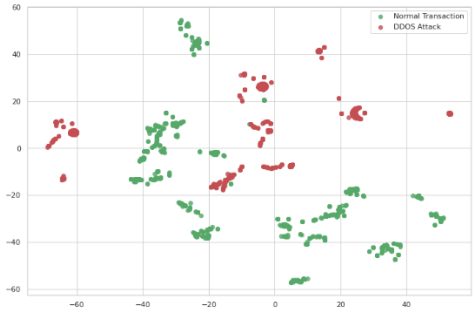

Fig 1.1 TNSE Plot for latent representation for DDoS (in red) and Normal Transactions (in green) Model4

Classification Report			
	Precision	Recall	F1 score
Benign	0.96	0.41	0.57
Attack	0.80	0.99	0.89
Macro Avg	0.88	0.70	0.73
Weighted Avg	0.85	0.82	0.79

Table 11 Classification report for final model (model 4)

The results were promising considering how the auto coder model was trained for 15,000 records. As part of future works, one may experiment with variational auto encoders to see if we obtain better latent representation during the unsupervised learning phase. Also, experiment with longer earning periods and more records for normal transaction to increase the accuracy, our dataset had very limited normal transactions to train the autoencoder models, but with the experiments conducted by our team, we observed that more the auto encoder model performed better with more data and longer training periods.

Model Explanation

Most of our models' accuracies are extremely high. One of the possible reasons may be that we made this into a binary classification problem, instead of a multiclass classification one, whereby each class is a DDoS attack type. By doing so, we reduced the complexity of the problem and improved the overall model performance. Another reason may be the huge class imbalance of our datasets, with most instances leaning towards 'Attack'. The models will have the tendency to better learn the characteristics of 'Attack' instances. Another possibility is that the dimensionality reduction done in the preprocessing steps were highly effective to minimize the noise in the datasets, allowing the models to learn all the required characteristics to differentiate 'Attack' and 'Benign' instances well.

Conclusion and Future Work

We chose to use precision as the overall evaluation metric to determine the best model; we look to minimize instances of false alarms (or false positives) as they have negative implications and costs to organizations investigating into possible DDoS attacks. The ANN model with deeper layers and dropout shows the highest precision (0.9985), and we conclude that it is the best model out of all.

In the future, we are interested in utilizing feature engineering on important features to further improve the models' performances. The most important features are related to the packets sent and abnormal flags within. These features will be closely monitored and engineered on to improve our models. We wish to look into applying our models in an actual network by extracting features from the network nodes. The detection engine can be a module or an application running on a centralized node that can be constantly fed with the data. The trained model can raise alarm to network admin for action when the anomaly is detected. By having the detection engine in a centralized node, the attack can be detected earlier.

References

Elsayed, M., Le-Khac, N.-A., Dev, S., & Jurcut, A. (2020). DDoSNet: A Deep-Learning Model forDetecting Network Attacks.

Gogna, A., & Majumdar, A. ((2016, October)). *Semi supervised autoencoder. In International Conference on Neural Information Processing (pp. 82-89).* Springer, Cham.