Detection of Cyber-attacks on Network Infrastructure using Machine Learning Models

Capstone Project for CodingNomads’ “Data Science and Machine Learning” course

Author: Delan Jayasuriya

Published: 2nd of February 2024

Abstract

Cyber-attacks on network infrastructure have become more and more prevalent and sophisticated over the years. An Intrusion Detection Systems (IDS) is often used to defend against such cyber threats. It is a network monitoring system that an IT department can deploy in order to detect such attacks based on captured network traffic/flow data. It can then send alerts to security incident responders to investigate and respond to potential threats.

In the past, most IDS employed some form of signature-based detection. They would have a database of known attack signatures – which had to be constantly updated whenever new attack signatures became known. They would often not detect unknown attack vectors. On the other hand, an anomaly-detection based IDS would attempt to identify network traffic that deviates from normal traffic behaviour. They are known to be more successful in detecting unknown attack vectors. Today, most IDS would employ a combination of signature-based detection and anomaly-based detection.

This capstone project was done as part of the CodingNomads’ “Data Science and Machine Learning” course. Its objective is to design and train a supervised ML (machine learning) model that can be used within an IDS to carry out anomaly-based detection to identify 13 network attack vectors. After evaluation of several multiclass classifiers, a LightGBM classifier was chosen as the most suitable for IDS with a F2 score of 100% for detecting 8 out of the 13 attack vectors. F2-macro score (which is the F2 score when averaged against all of the attack vectors) was measured to be 91%.

Introduction

A good intrusion detection system will accurately detect cyber-attacks and malicious network activity and promptly alert IT/security incident teams. An IDS that raises too many inaccurate/false alarms (ie false positives) might lead to benign network traffic being unnecessarily blocked – which could frustrate employees and impact on their productivity. On the other hand, an IDS that results in too many false negatives will result in some network intrusion attempts that might not get picked up in time before they can cause serious damage to an organisation’s network, reputation and data security.

As part of this capstone project, a machine learning model was successfully built which can detect 13 different types of cyber-attacks – all in the midst of typical network traffic that is predominately benign. It is able to do this with a high degree of precision and recall across majority of those attack vectors – with an overall F2-macro score of 0.91 on the Testing Set. This includes:

- 100% success rate in detecting the following attack vectors: Botnet, several types of Distributed Denial-of-Service attacks, SSH brute force attacks, FTP brute force attacks and all types of Denial-Of-Services attacks (eg GoldenEye, Hulk and Slowloris).
- Over 91% success rate (F2-macro score) for the following attack vectors: XSS (Cross-site scripting) brute force attacks and LOIC Distributed Denial-of-Service attacks.
- Despite relatively low F2 scores for the following attack vectors, it has achieved high recall rates of over 92%: brute force attacks over HTTP and infiltration attacks.

Dataset

Training and testing of this ML model was performed on a dataset known as the "CSE-CIC-IDS2018 dataset" which was produced as part of a collaboration project between the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC). It can be downloading from this link: https://www.unb.ca/cic/datasets/ids-2018.html

This dataset contains 16 million samples with 80 features on each sample. Each data point is captured network data that originated within an AWS computing platform that was configured to replicate a common LAN network topology which includes:

- 5 subnets, one for each department of a fictitious organisation/company
- 420 desktop machines and 30 servers (combinations of Windows, Ubuntu and Windows Server machines)
- Attacking infrastructure which consists of 50 machines

The dataset captures network traffic which is mostly benign network flow data together with a small percentage of network attacks captured over a 10-day period. It is meant to replicate a scenario where a corporate network is being used by its employees (benign traffic) while it is being targeted by cybercriminals. The dataset includes a "Label" column which identifies traffic as being either benign or one of the 14 different attack classes under the following distribution:

Class Label	**Count of Samples**	**Sample %**
Benign	13,484,708	83.070%
Bot	286,191	1.763%
Brute Force -Web	611	0.004%
Brute Force -XSS	230	0.001%
DDOS attack-HOIC	686,012	4.226%
DDOS attack-LOIC-UDP	1,730	0.011%
DDoS attacks-LOIC-HTTP	576,191	3.550%
DoS attacks-GoldenEye	41,508	0.256%
DoS attacks-Hulk	461,912	2.846%
DoS attacks-SlowHTTPTest	139,890	0.862%
DoS attacks-Slowloris	10,990	0.068%
FTP-BruteForce	193,360	1.191%
Infiltration	161,934	0.998%
SQL Injection	87	0.001%
SSH-Bruteforce	187,589	1.156%
Grand Total	**16,232,943**	**100%**

Table 1: Distribution of target classes in the dataset

The network packets and flow data were captured through a tool called CICFlowMeter-V3 which extracts 80 features based on that network data. Some of the main features include:

- Duration of network flow
- Number of packets
- Destination port
- Length of the packets
- Min/max/mean/std of the packet length - in both forward direction and backward direction of the network flow
- Time between two packets sent in the flow
- Number of bytes
- Number of bytes in the header
- TCP flags found in the header of each packet
- Download and upload ratio…etc

Types of network attacks captured

Among benign traffic, the dataset captures 7 different attack categories which includes:

- **Brute-force attacks**: Several tools based on FTP and SSH were used to conduct brute-force attacks and password cracking. A large dictionary that contains 90 million words were used as the listing of passwords.
- **Heartbleed attacks**: A tool was used to scan the network for systems that are vulnerable for the Heartbleed bug and then to exploit that bug and exfiltrate data.
- **Botnet**: Machines were infected with 2 different types of botnets (Zeus, which is a Trojan horse malware package that are used in phishing schemes and Ares, which is an open-source botnet that can capture screenshots and key logging).
- **Denial-of-Service (DoS)**: use the Slowloris tool which allows a single machine to take down a web server
- **Distributed Denial-of-Service (DDoS)**: Based on the HOIC tool which can attack as many as 256 URLs at the same time
- **Web Attacks**: A PHP & MySQL based vulnerable website is attacked using XSS (cross-site scripting).
- **Infiltration**: Simulates the scenario where a user receives a malicious document through email which opens a backdoor on the victim's computer to allow it to carry out various attacks on the network.

There is a total of 14 types of attack classes across those 7 categories.

Related work

Several other research papers have also used this same "CSE-CIC-IDS2018" dataset for the purpose of evaluating algorithms for anomaly-based detection of network attacks. Examples of such work include:

- Zuech R, Hancock J and Khoshgoftaar TM, "Detecting web attacks using random undersampling and ensemble learners", 2021
    - Uses Area Under the Receiver Operating Characteristic Curve (AUC) and Area Under the Precision-Recall Curve (AUPRC) as evaluation metrics.
    - Claims to be the "first to apply random undersampling techniques to web attacks from the CSE-CIC-IDS2018 dataset while exploring various sampling ratios."

    - Explores the effect of various random undersampling ratios on several classifiers including LightGBM, XGBoost and CatBoost and Naive Bayes
- Huancayo Ramos KS, Sotelo Monge MA, Maestre Vidal J. “Benchmark-based reference model for evaluating botnet detection tools driven by traffic-flow analytics”. 2020
    - Looks at only the Botnet target class in both the CSE-CIC-IDS2018 dataset and the Isot Http Botnet Dataset
    - Precision and Recall are the primary evaluation metrics
    - Concludes that Random Forest, Decision Tree and K-Neighbors models yield a classification accuracy close to 100% for detecting the Botnet class
- Atefinia R, Ahmadi M. “Network intrusion detection using multi-architectural modular deep neural network”. 2020
    - Proposes a novel modular deep neural network model
    - Evaluation metrics include F1 and Accuracy
    - Claims F1 score of 1.0 on several attack vectors
- Serinelli BM, Collena A, Nijdam NA. “On the analysis of open source datasets: validating IDS implementation for well-known and zero day attack detection”. 2021:
    - Evaluates 12 other open-source datasets besides the CSE-CIC-IDS2018 dataset
    - Claims that those 12 datasets have numerous flaws with the data and is not sufficient for anomaly-based IDS. CSE-CIC-IDS2018 dataset is better suited.
    - Compares performance of both python solution vs R. The python models were found to be faster
    - Training was primarily done using XGB, Random Forest and SVM
    - Evaluation metrics used are Accuracy, FAR (False alarm rate) and UA (Undetected Attack) rate.
    - No mention of the impacts of imbalanced datasets
    - Their proposed prediction model has an accuracy of 98% to 99% against most attack vectors. However there doesn’t seem to be any acknowledgement of the drawbacks of using accuracy as an evaluation metric on heavily imbalanced datasets.

The work done as part of this project differs from any of the above-mentioned related works in terms of:

- The evaluation metrics being used: None of the other related works have proposed prioritising recall over precision for IDS. And nor have they advocated the benefits of the use of F2-macro and MCC as evaluation metrics
- The elimination of labelling errors in the dataset: Several papers have picked up some minor errors with the dataset, but none of them seem to have detected a major labelling error of one of the target classes. Refer to “Exploratory data analysis & data cleansing” section below for more details.

Challenges of this project

- Large dataset: 16 million data points with 80 features each. The dataset consists of 6.4GB of files.
- Highly imbalanced dataset: Only 17% of the dataset has attack vectors. Some attack vectors such as SQL Injection only had a total of 87 samples, though the size of the dataset is over 16 million samples. That is, just 0.001% of the dataset!
- Several data issues were detected on the dataset. Eg There were 139,890 samples that were found to have mis-classified target labels. These had to be removed prior to any training

Methodology

Several supervised machine learning algorithms were trained and evaluated on the CSE-CIC-IDS2018 dataset with the objective of selecting a single multiclass classification model that can detect all 14 types of network intrusion attempts as part of an IDS.

These algorithms were built on a Python implementation running on an Anaconda distribution (Windows environment). Version details of packages used:

- Python version 3.11.5
- Scikit-learn machine learning library version 1.2.2
- Anaconda version 2.5.2
- Pandas version 2.1.4
- Numpy version 1.26.2

Process and methodology that ultimately lead to the optimal supervised machine learning model:

Data preparation

- As the dataset was heavily imbalanced and containing a high percentage of benign samples, it was deemed necessary to cull significant amounts of the benign samples from each file - so that dataset is more manageable. This was done by stratifying all the benign samples by the “Protocol” and “Dst Port” features and then reducing up to 80% of samples within each stratum. This stratification ensures that the reduced dataset will still retain key data samples under each stratum.
- The Timestamp column is then dropped – as it is not required for this project, given we are not doing any time-series based modelling. IP address related columns were also dropped – as we want the model to detect intrusion attempts regardless of source and destination IP addresses.
- 8 features that had all-zero data were also dropped

Exploratory data analysis & data cleansing

- Examination of samples that have null/NaN values have revealed 59,215 samples that had a value of 0 in the “Flow Duration” feature. Given the flow duration is measured in microseconds, we expect every network flow to have a non-zero value. Most likely some error might have occurred during the network flow capture which would have caused the CICFlowMeter tool to report this feature as 0. A decision was made to remove these 59,215 samples – this also removed all samples that had null/NaN data (as those NaN values were most likely caused as a result of division-by-zero errors stemming from the “Flow Duration” feature being 0)
- Further exploration of data lead to the discovery of 139,890 samples that were labelled as “DoS attacks-SlowHTTPTest”, but looks very similar to samples that have been labelled with the “FTP-BruteForce” target class. It was also noted that the Destination Port feature is "21" for both of these classes. Port 21 is expected for FTP-BruteForce attacks (as port 21 is a well-known port for FTP). However, for 'DoS attacks-SlowHTTPTest', the expected port is either 80 (for HTTP attacks) or port 443 (for HTTPS attacks). Hence it is highly likely that the 'DoS attacks-SlowHTTPTest' samples were created with the incorrect label. Most likely those samples are actually FTP-Bruteforce attacks, but incorrectly labelled as 'DoS attacks-SlowHTTPTest'. A decision was made to remove all samples with the “DoS attacks-SlowHTTPTest” target label from the model.

Creation of training and testing sets

- The dataset was split into training and testing sets with stratification on the target class. The training set was allocated 75% of stratified samples, while the testing set was allocated the remaining 25% samples
- The testing set is then persisted to a file and not used or referred to until the final stages of model evaluation. This minimizes the possibility of train-test contamination.
- Further exploration was done on the training set:
    - Several outliers were observed on the training set. The “Flow Byts/s” feature had 30 outliers. As these outliers could impact the model training, these 30 samples were removed.
    - All columns are numerical except for these 2 categorical features: “Protocol” and “Dst Port”
- 25% of the training dataset seems to have duplicated data. All duplicates of these classes were removed: Benign & DDOS attack-HOIC. The remaining duplicated data was then stratified by the target class, and 80% of those duplications were deleted within each stratum.
- The benign and DDoS attacks-LOIC-HTTP classes were further culled through random sample deletions - so that it’s sample counts would come closer to that of other majority classes. This would not only reduce the size of the training set significantly (which would aid in reducing the time required to train the model), but would make the dataset more balanced.
- The breakdown of classes in the training and testing sets are now as follows:

Label	**Training Set**	**Training Set**	**Testing Set**	**Testing Set**
	Distribution	**%**	**Distribution**	**%**
DoS attacks-Hulk	164,962	16.499%	115,478	7.704%
DDOS attack-HOIC	156,456	15.648%	171,503	11.441%
DDoS attacks-LOIC-HTTP	151,250	15.127%	144,048	9.610%
Bot	129,813	12.983%	71,548	4.773%
Benign	121,965	12.198%	847,255	56.521%
Infiltration	120,476	12.049%	40,160	2.679%
SSH-Bruteforce	84,521	8.453%	46,897	3.129%
DoS attacks-GoldenEye	31,131	3.114%	10,377	0.692%
FTP-BruteForce	29,045	2.905%	48,339	3.225%
DoS attacks-Slowloris	8,243	0.824%	2,747	0.183%
DDOS attack-LOIC-UDP	1,298	0.130%	432	0.029%
Brute Force -Web	458	0.046%	153	0.010%
Brute Force -XSS	173	0.017%	57	0.004%
SQL Injection	65	0.007%	22	0.001%

Table 2: Breakdown of target classes in Training Set vs Testing Set

Evaluation metrics

Before a classifier can be evaluated and chosen, it is necessary to determine which evaluation metric can be used to compare the performance of each model/classifier. To effectively account for the known use-cases of an IDS, and to account for the heavily imbalanced nature of the attack vectors, the following evaluation metrics were proposed:

- **Precision** – In the context of an IDS, this answers the question: “What proportion of positive identifications were actually true network intrusion attempts?”. Precision is defined as:

$$\text{Precision} = \frac{TP}{(\text{TP} + \text{FP})}$$

where the two variables are quadrants of the Confusion Matrix: TP represents the True Positive quadrant and FP represents the False Positive quadrant.

- **Recall** – This answers the question "Out of all network intrusion attempts, how many were successfully detected by the IDS?". Recall is defined as:

$$\text{Recall} = \frac{TP}{(\text{TP} + \text{FN})}$$

where the two variables are quadrants of the Confusion Matrix: TP represents the True Positive quadrant and FN represents the False Negative quadrant.

- **F2 macro-averaged score** – This is defined as the average of all the F2 scores of each class, where F2 is defined as:

$$F_2 = 5.\frac{Precision.Recall}{(4.Precision) + Recall}$$

The F2 score was chosen over the F1 score due to the additional weighting it provides towards Recall.

  - Though both Precision and Recall are important attributes of an IDS, higher Recall is considered to be a more critical and sought-after attribute of an effective IDS.
  - If the Precision metric of an IDS is too low, that would indicate that a larger portion of alerts generated by the IDS could be false alarms. This might result in employee frustration and impact to their productivity. However, various strategies can be used by the IDS to minimise false alarms (eg only alert once per hour, and log all other alerts for later analysis…etc)
  - If the Recall metric of an IDS is too low, it would indicate that relatively higher amounts of malicious network activity could be going undetected. This could lead to data and/or privacy breaches that could cause monetary and reputational impact to the organisation. Hence the Recall metric of an IDS would be more critical than the Precision metric
  - Hence the F2 score would be ideal in emphasising Recall over Precision – while retaining some stake in Precision as well

- **Matthews Correlation Coefficient (MCC)** – This metric captures all 4 quadrants of the Confusion Matrix in its definition (whereas F2 score only captures 3 quadrants). This metric is defined as:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

where the variables are quadrants of the Confusion Matrix: TP represents the True Positive quadrant, FN represents the False Negative quadrant, FP represents the False Positive quadrant and TN represents the True Negative quadrant. Note that for multiclass classifiers, the calculation of the metric is more complex than the above equation. Hence, we rely on the scikit-learn's matthews_corrcoef function which correctly calculates MCC for multiclass classifiers.

Essentially, the 2 primary evaluation metrics used throughout this project are F2-macro score and the MCC score. Though using a single evaluation metric would have been sufficient, a decision was made to use both F2 and MCC together when evaluating the performance of classifiers, as two independently calculated metrics together will provide a more convincing view of performance of the model.

Evaluating potential classifiers

- Before a suitable classifier can be evaluated, we need to transform the training set to a state which would make it compatible with a large selection of classifiers. The following transformations were done to that effect:
  - Categorical columns were encoded using OneHotEncoder
  - All numerical columns were scaled using StandardScaler
  - All target class labels were converted to numerical values using a mapping function (which works similar to the LabelEncoder function)
  - The training set was undersampled so the majority classes would only have 10,000 samples each. This would make the training set much less imbalanced and would require less time to train each classifier.
- The scikit-learn's "DummyClassifier" was used as the baseline model. It was trained using the "stratified" strategy, which means predictions were done by blindly guessing the classes at the same ratio as was found in the training set.
- 12 classifier models were evaluated using scikit-learn's cross_val_score function, which was configured to evaluate using StratifiedKFold cross-validation and the two chosen metrics: F2 macro and MCC score. The time taken to train and evaluate each classifier is also recorded (as the time performance of the classifier is also an important aspect of an IDS). Results of this testing:

Classifier Name	F2 macro	MCC	Time Taken (seconds)
DummyClassifier	0.07	0.00	2.86
DecisionTreeClassifier	0.94	0.96	4.85
RandomForestClassifier	0.95	0.97	13.9
AdaBoostClassifier	0.95	0.97	173.27
RidgeClassifier	0.76	0.85	1.95
LinearSVC	0.85	0.95	191.21
LogisticRegression	0.85	0.95	25.95
KNeighborsClassifier	0.91	0.97	15.43
SVC	0.75	0.84	2546.62
GaussianNB	0.72	0.84	2.01
XGBClassifier	0.95	0.97	182.37
LGBMClassifier	0.96	0.98	31.01
CatBoostClassifier	0.94	0.97	1373.55

Table 3: Results of preliminary evaluation of classifiers. Evaluation was done on a heavily undersampled training set in which the majority classes have approx. 10,000 samples each.

- As can be seen, all models performed significantly better than the DummyClassifier.
- The highest F2-macro and MCC scores were obtained by the LGBMClassifier. Its time taken was also very impressive. The RandomForestClassifier has come a close-second.

- Based on the above results, the LGBMClassifier was chosen as the ideal candidate for training the IDS.

More about the chosen classifier:

- LGBMClassifier is a model that is imported from Light GBM (Light Gradient Boosting Machine) which is an open-source distributed gradient-boosting framework for machine learning developed by Microsoft.
- It is based on a gradient boosting framework that uses decision trees. But unlike other decision tree-based algorithms, Light GBM uses a histogram-based approach to "bin" continuous features. This enables Light GBM to use less memory, thereby enabling it to operate much faster than most other boosting algorithms.

Experiments done

Having decided on a classifier, several experiments were then carried out to assess the impact of various dataset balancing techniques, dimensionality reduction techniques and model tuning to determine the optimal parameters of the LGBMClassifier.

Summary of experiments conducted:

Dataset balancing

Heavily imbalanced datasets can significantly reduce the accuracy of many machine learning algorithms. LGBMClassifier is no exception. There are several undersampling and oversampling techniques (eg undersampling of majority classes to bring down their sample size followed by oversampling of minority classes to bring their sample sizes up) can be used to transform the imbalanced dataset to a balanced state during training.

The following experiments were done to assess the various strategies for resolving for the class imbalance issue on the dataset (see table below for results):

1. Using full (imbalanced) training set without any undersampling/oversampling or class weights. This is the baseline for our assessment
2. Using full (imbalanced) training set without any undersampling/oversampling. Class weights are used on the classifier
3. Using RandomUnderSampler to first reduce the sample sizes of majority classes to 50000. Then use the RandomOverSampler to increase the sample sizes of minority classes to 50000.
4. Using NearMiss (version 2) to first reduce the sample sizes of majority classes to 50000. Then use SMOTE to increase the sample sizes of minority classes to 50000.
5. Using RandomUnderSampler to first reduce the sample sizes of majority classes to 50000. Then use the SMOTE to increase the sample sizes of minority classes to 50000.
6. Using RandomUnderSampler to first reduce the sample sizes of majority classes to 50000. Then use the BorderlineSMOTE to increase the sample sizes of minority classes to 50000.
7. Using NearMiss (version 1) to first reduce the sample sizes of majority classes to 50000. Then use BorderlineSMOTE to increase the sample sizes of minority classes to 50000.
8. Using NearMiss (version 2) to first reduce the sample sizes of majority classes to 50000. Then use BorderlineSMOTE to increase the sample sizes of minority classes to 50000.
9. Partially balancing the dataset using RandomUnderSampler to reduce the sample sizes of majority classes to 50000. Then use class weights on the classifier

Results of the experiments:

Strategy for balancing dataset	Training Set			Testing Set	
	Training Set size	F2-Macro score	Matthews Correlation Coefficient	F2-Macro score	Matthews Correlation Coefficient
1: Imbalanced dataset and without class weights	999856	0.10	0.12	0.09	0.03
2: Imbalanced dataset. Using class weights on the classifier	999856	0.98	0.98	0.88	0.91
3: RandomUnderSampler followed by RandomOverSampler	700000	0.99	0.98	0.90	0.90
4: NearMiss (version 2) followed by SMOTE	700000	0.98	0.98	0.68	0.77
5: RandomUnderSampler followed by SMOTE	700000	0.98	0.98	0.89	0.90
6: RandomUnderSampler followed by BorderlineSMOTE	679045	0.98	0.98	0.91	0.90
7: NearMiss (version 1) followed by BorderlineSMOTE	679045	0.40	0.41	0.31	0.30
8: NearMiss (version 2) followed by BorderlineSMOTE	679045	0.98	0.98	0.60	0.63
9: RandomUnderSampler on majority classes and then class weights on the Classifier	470408	0.98	0.98	0.87	0.91

According to these results, strategy 6 (the use of RandomUnderSampler followed by BorderlineSMOTE) seems to produce the best outcome for model training, with strategy 3 coming a close-second.

Hence the combination of RandomUnderSampler and BorderlineSMOTE will be used to resize and condition the training set going forward.

Model Tuning

For model tuning, StratifiedKFold was used to split the training set into 5 folds. Cross-validation scores (based on F2-macro score) were obtained by testing against each fold. This enables the training set to be recycled for hyper-tuning of the classifier.

Hyper-tuning was done using Optuna, which is an automatic hyperparameter tuning framework. Optuna was chosen over other commonly used hyper-tuning classes such as GridSearch or RandomizedSearch due to its performance in quickly converging when you have a large number of parameters to tune.

The following parameters of the Light GBM classifier were tuned using Optuna:

- "n_estimators": Range: 20 to 250
- "learning_rate": Range: 0.03 to 0.4
- "num_leaves": Range: 20 to 250
- "max_depth": Range: 3 to 15
- "max_bin": Range 50 to 450, step=50

- "boosting_type": Either 'gbdt' or 'dart'
- "min_data_in_leaf": Range: 200 to 10000, step=100
- "lambda_l1": Range: 0 to 100
- "lambda_l2": Range 0 to 100, step=5
- "min_gain_to_split": Range 0 to 15
- "bagging_fraction": Range 0.2 to 0.95
- "bagging_freq": Range 0 to 2
- "feature_fraction": Range 0.2 to 0.95, step=0.1

After repeating this hyper-tuning several times, the following ideal parameters were found that resulted in an F2-macro score of `0.98` on the cross-validations of the training set:

```
learning_rate: 0.1158766943612183
num_leaves: 152
max_bin: 400
boosting_type: gbdt
bagging_fraction: 0.8074313468720595
bagging_freq: 0
```

Required training set size

Learning Curves were used to plot the classifier's performance when trained using training sets of various sizes. A StratifiedKFold of 7 splits was passed to the scikit-learn's learning_curve function to evaluate the F2-macro score of the classifier when trained using training sets that range from as 94k to 800k. Resulting plot:

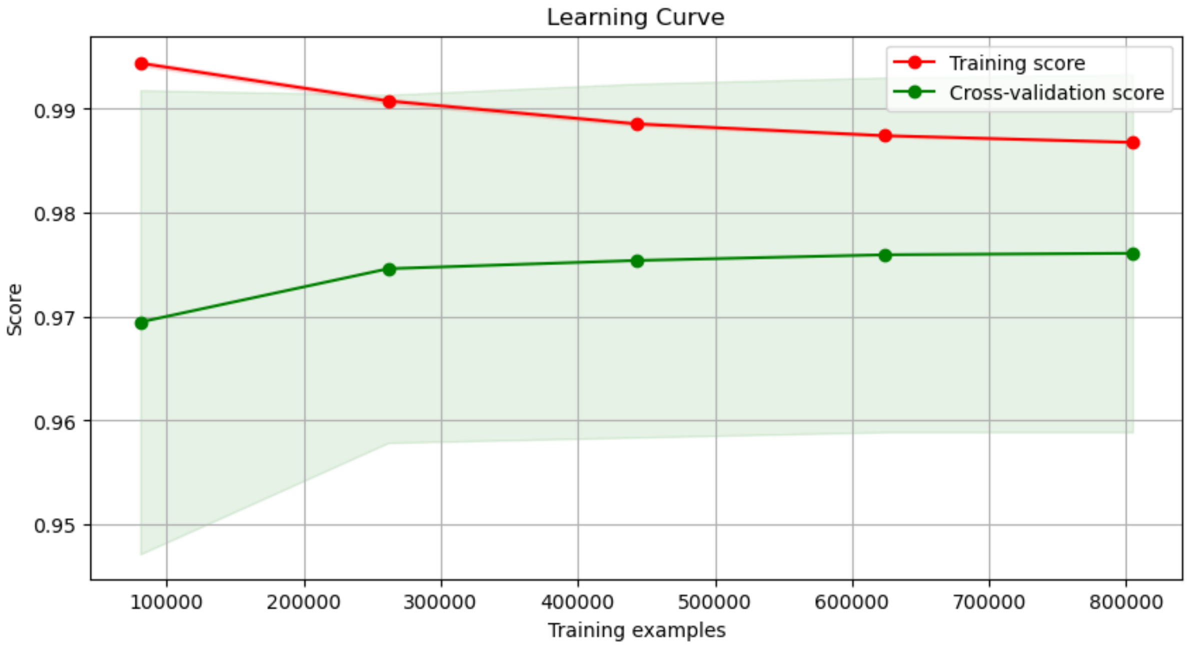


Figure 1: Learning curve to assess the ideal training set size

The plot indicates that the optimum training set size should be about 600,000 samples for this classifier (as the cross-validation score seems to peak according to the above plot when the training set size is between 600,000 and 700,000). This implies that each class should have roughly 43k samples (600k/14 classes ≈ 43k samples per class) for optimal training.

Principal Component Analysis

The dataset has 77 features. The dimensionality of the dataset may not be considered as being very large compared to many modern datasets. However, if the dimensionality of the dataset can be significantly reduced through techniques such as Principal Component Analysis (PCA), it might lead to less overfitting of the model.

Two experiments were done to assess the impact of PCA on the chosen classifier: PCA where n_components is set to 90% is compared to that of 99%:

PCA explained variance %	Resulting number of PCA components	Training Set			Testing Set	
		Training Size	F2 Macro score	Matthews Correlation Coefficient	F2 Macro score	Matthews Correlation Coefficient
90%	16	700000	0.98	0.98	0.85	0.88
99%	27	700000	0.98	0.98	0.87	0.88

Table 4: Accessing the impact of Principal Component Analysis on the chosen classifier

The above results indicate that the performance of the model doesn't significantly drop when the feature set is reduced from 77 to 16. This reduced model dimensionality might be beneficial in a productionised model, as the model might perform more efficiently and provide a quicker response.

However, for this project we are exploring what is the maximum possible F2-macro and MCC scores we can obtain. Also, PCA reduces model explainability (as all features are no longer representing real-world measures). Hence, we will not be using PCA going forward in this project.

Using SHAP analysis to reduce feature set

SHAP (SHapley Additive exPlanations) analysis was done to get an understanding of which features are of primary importance to the chosen classifier. The top 20 features used to detect the 14 target classes are:

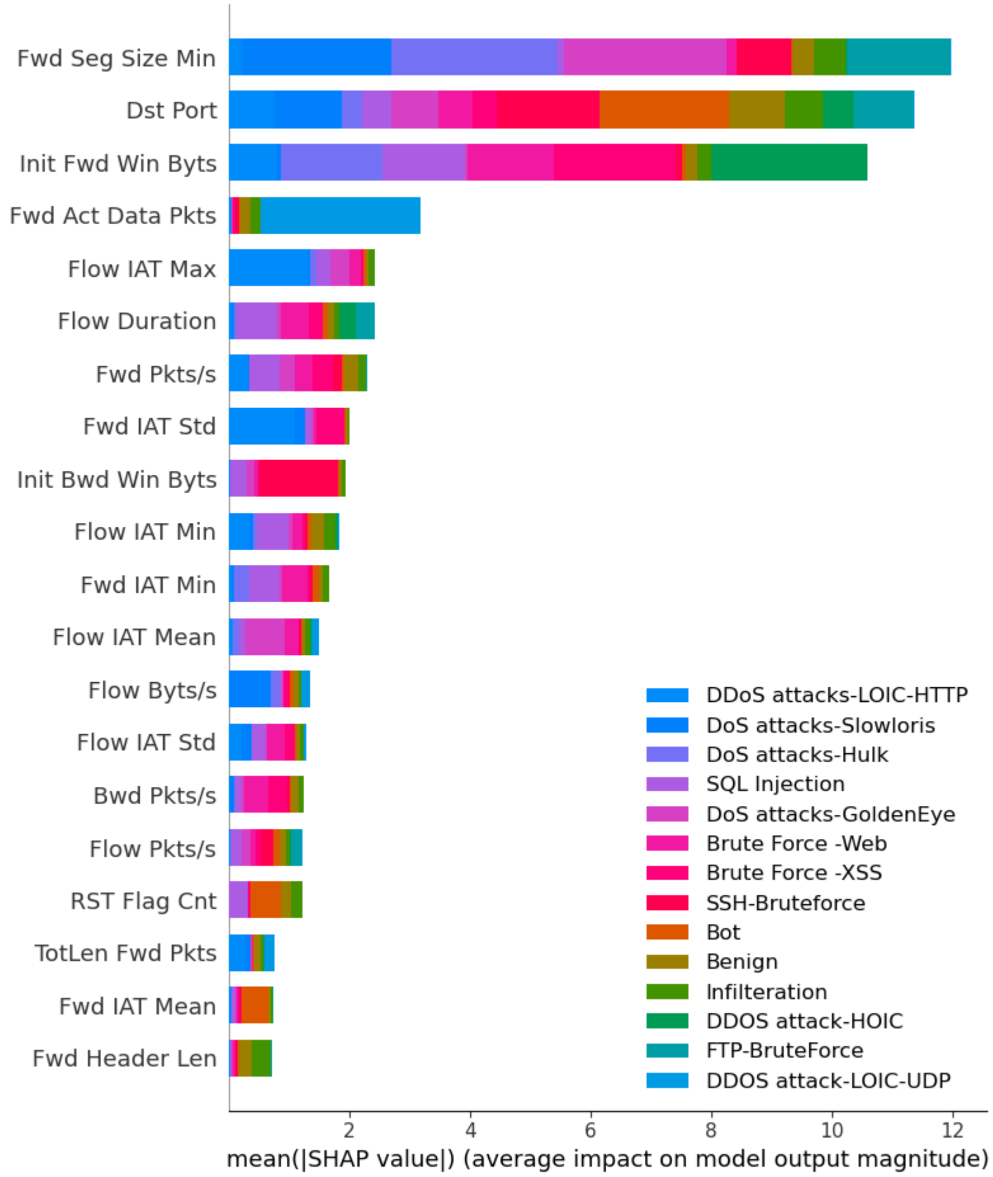


Figure 2: Summary Plot of SHAP values

Consequently, SHAP analysis was also used to determine which features are least important to the model. It was hoped that by dropping those features that are not considered to be very impactful to the model, there would be less overfitting.

The following features we found to be least important to the Light GBM classifier:

- Bwd Seg Size Avg
- CWE Flag Count,
- Fwd Seg Size Avg
- SYN Flag Cnt
- Subflow Bwd Pkts
- Subflow Fwd Pkts

After dropping those features, the overall performance of the model increased slightly – from an F2 macro score of 0.912 to 0.914. This marginal increase in performance may not be statistically significant. Nonetheless, it proved that reducing the feature set has potential to further improve the performance of the model by reducing overfitting.

Results and Analysis

The tuned LightGBM classifier was trained using a balanced training set – which was obtained by using the RandomUnderSampler followed by BorderlineSMOTE. This resulted in a balanced dataset where each class had about 50,000 samples.

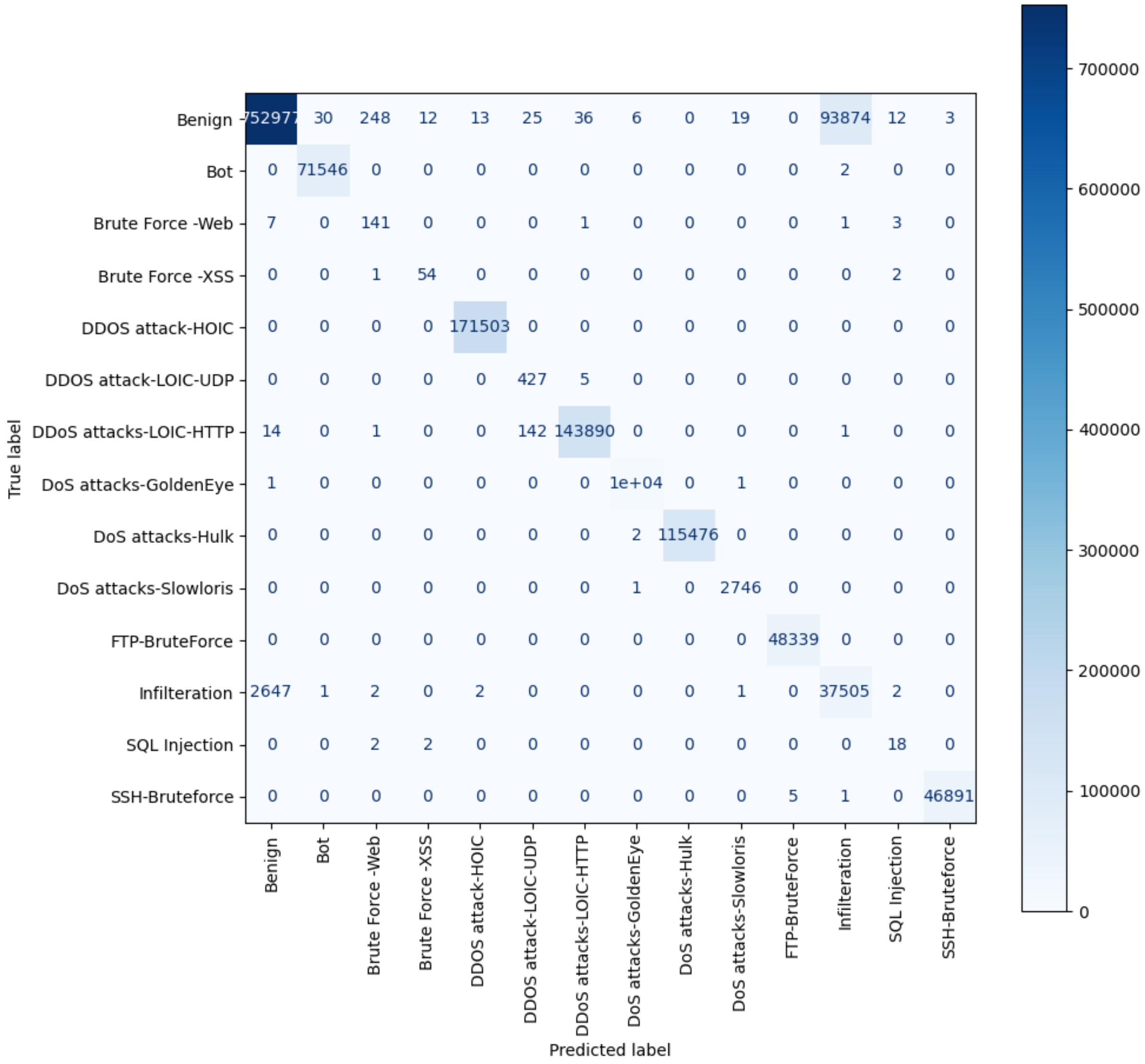
When the testing set (which had 1,499,016 samples) was passed to the model, it was successfully able to detect a large percentage of network attacks with an overall F2-macro score of 0.914 and a MCC score of 0.91.

The evaluation metrics for each class:

Target Class	**Precision**	**Recall**	**F2-score**	**Support**
Benign	1.00	0.89	0.91	847,255
Bot	1.00	1.00	1.00	71,548
Brute Force -Web	0.36	0.92	0.70	153
Brute Force -XSS	0.79	0.95	0.91	57
DDOS attack-HOIC	1.00	1.00	1.00	171,503
DDOS attack-LOIC-UDP	0.72	0.99	0.92	432
DDoS attacks-LOIC-HTTP	1.00	1.00	1.00	144,048
DoS attacks-GoldenEye	1.00	1.00	1.00	10,377
DoS attacks-Hulk	1.00	1.00	1.00	115,478
DoS attacks-Slowloris	0.99	1.00	1.00	2,747
FTP-BruteForce	1.00	1.00	1.00	48,339
Infiltration	0.29	0.93	0.64	40,160
SQL Injection	0.49	0.82	0.72	22
SSH-Bruteforce	1.00	1.00	1.00	46,897

Table 5: Performance of the final tuned model against the Testing Set

Confusion Matrix:



True label \ Predicted label	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	752977	30	248	12	13	25	36	6	0	19	0	93874	12	3
Bot	0	71546	0	0	0	0	0	0	0	0	0	2	0	0
Brute Force -Web	7	0	141	0	0	0	1	0	0	0	0	1	3	0
Brute Force -XSS	0	0	1	54	0	0	0	0	0	0	0	0	2	0
DDOS attack-HOIC	0	0	0	0	171503	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	427	5	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	14	0	1	0	0	142	143890	0	0	0	0	1	0	0
DoS attacks-GoldenEye	1	0	0	0	0	0	0	1e+04	0	1	0	0	0	0
DoS attacks-Hulk	0	0	0	0	0	0	0	2	115476	0	0	0	0	0
DoS attacks-Slowloris	0	0	0	0	0	0	0	1	0	2746	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	48339	0	0	0
Infilteration	2647	1	2	0	2	0	0	0	0	1	0	37505	2	0
SQL Injection	0	0	2	2	0	0	0	0	0	0	0	0	18	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	5	1	0	46891

Figure 3: Confusion Matrix of the final model against the Testing Set

Explanation of outcome

- It correctly detected 100% of the samples under the following class labels: Bot, DDOS attack-HOIC, DDoS attacks-LOIC-HTTP, DoS attacks-GoldenEye, DoS attacks-Hulk, DoS attacks-Slowloris, FTP-BruteForce and SSH-Bruteforce
- The worse F2-macro score of 0.64 was obtained for the Infiltration class. This was expected given infiltration attacks often mimic benign traffic. The Confusion Matrix reveals that majority of the predicted Infiltration traffic was actually benign ie the model had low precision (29%) when it comes to detecting Infiltration attacks. However, nearly all of the actual Infiltration attacks were successfully detected ie the model has a high recall of 93% for Infiltration attacks
- The next worse F2-macro score was obtained for the detection of Brute Force -Web and SQL Injection attacks. This is most likely due to the very low number of samples of these 2 classes that were in the dataset – less than 500 samples out of 16 million data points for both of these classes. It would be possible to obtain higher F2 scores for these classes if a training dataset with more of these samples can be obtained.

- If these 3 classes (Brute Force -Web, SQL Injection and Infiltration) are excluded, the F2-macro score for the remaining 11 classes would be 0.98 – which would be considered as a high degree of accuracy for an IDS

Training this model using the balanced training set took 35 seconds. Testing 1000 samples in the training set took only 30 milliseconds. This is an acceptable level of speed performance for an IDS in production – especially considering that the model hasn't been tuned for speed performance (not in scope of this project).

Conclusion

The LightGBM is a highly suitable multi-class supervised classifier that is able to successfully detect 13 different types of attack vectors with an overall F2-macro score of 0.91 and a MCC (Matthews Correlation Coefficient) of 0.91

Given the "CSE-CIC-IDS2018 dataset" is highly imbalanced, it was necessary to first prepare the training set prior to training the classifier. This includes undersampling the majority classes and oversampling the minority classes. With correct training and hyper-tuning, the LightGBM classifier would be considered as a highly performant solution for an IDS.

Future work

The following can be undertaken to further optimise the model:

- When evaluating potential classifiers, a better approach might have been to use cross_val_score function to first shortlist up to 3 models (instead of arriving at a single model as was done in this project). Then hyper-tuning can be done on all 3 of those shortlisted models before selecting the most optimum model. This could potentially give a better result as the chosen model is picked among a pool of already hyper-tuned models
- The scope of this capstone project was limited to machine learning models such as Random Forest, LightGBM, XGB…etc. However, it would be interesting to examine whether deep-learning based models could further improve the outcome.

Acknowledgement

Thanks to Gilad Gressel from CodingNomads (https://codingnomads.co) for being such an inspiring instructor and for running the brilliant course "Data Science + Machine Learning"

The data for this project was obtained from: https://registry.opendata.aws/cse-cic-ids2018/

It is known as the "Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)" which was produced through a collaborative project between the Communications Security Establishment (CSE) and The Canadian Institute for Cybersecurity (CIC).

Appendix A – Description of Features

Listing of features found in the datasets:

Feature Name	Description
Dst Port	Destination port number
Protocol	Transport protocol
Timestamp	Time of arrival of initial packet at destination
Flow Duration	Duration of the flow in Microsecond
Tot Fwd Pkts	Total packets in the forward direction

Tot Bwd Pkts	Total packets in the backward direction
TotLen Fwd Pkts	Total size of packet in forward direction
TotLen Bwd Pkts	Total size of packet in backward direction
Fwd Pkt Len Min	Minimum size of packet in forward direction
Fwd Pkt Len Max	Maximum size of packet in forward direction
Fwd Pkt Len Mean	Mean size of packet in forward direction
Fwd Pkt Len Std	Standard deviation size of packet in forward direction
Bwd Pkt Len Min	Minimum size of packet in backward direction
Bwd Pkt Len Max	Maximum size of packet in backward direction
Bwd Pkt Len Mean	Mean size of packet in backward direction
Bwd Pkt Len Std	Standard deviation size of packet in backward direction
Flow Byts/s	Number of flow bytes per second
Flow Pkts/s	Number of flow packets per second
Flow IAT Mean	Mean time between two packets sent in the flow. IAT = inter-arrival time
Flow IAT Std	Standard deviation time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Tot	Total time between two packets sent in the forward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Tot	Total time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Len	Total bytes used for headers in the forward direction
Bwd Header Len	Total bytes used for headers in the backward direction
Fwd Pkts/s	Number of forward packets per second
Bwd Pkts/s	Number of backward packets per second
Pkt Len Min	Minimum length of a packet
Pkt Len Max	Maximum length of a packet
Pkt Len Mean	Mean length of a packet
Pkt Len Std	Standard deviation length of a packet
Pkt Len Var	Variance length of a packet
FIN Flag Cnt	Number of packets with FIN
SYN Flag Cnt	Number of packets with SYN

RST Flag Cnt	Number of packets with RST
PSH Flag Cnt	Number of packets with PUSH
ACK Flag Cnt	Number of packets with ACK
URG Flag Cnt	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Cnt	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Pkt Size Avg	Average size of packet
Fwd Seg Size Avg	Average size observed in the forward direction
Bwd Seg Size Avg	Average number of bytes bulk rate in the backward direction
Fwd Byts/b Avg	Average number of bytes bulk rate in the forward direction
Fwd Pkts/b Avg	Average number of packets bulk rate in the forward direction
Fwd Blk Rate Avg	Average number of bulk rate in the forward direction
Bwd Byts/b Avg	Average number of bytes bulk rate in the backward direction
Bwd Pkts/b Avg	Average number of packets bulk rate in the backward direction
Bwd Blk Rate Avg	Average number of bulk rate in the backward direction
Subflow Fwd Pkts	The average number of packets in a sub flow in the forward direction
Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Pkts	The average number of packets in a sub flow in the backward direction
Subflow Bwd Byts	The average number of bytes in a sub flow in the backward direction
Init Fwd Win Byts	The total number of bytes sent in initial window in the forward direction
Init Bwd Win Byts	The total number of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Label	Identifies traffic as being either benign or one of the 7 different attack scenarios that is in scope for this project

Table 6: List of dataset features and their definition