

Comparative Analysis of Various Machine Learning Techniques Applied Towards Intrusion Detection in Computer Networks

Ghaniyyat Bolanle Balogun^{1*}, Olugunna Samuel Babade¹, Joseph Bamidele Awotunde¹, Muhideen Abdulraheem¹, Idowu Dauda Oladipo¹

¹Department of Computer Science, University of Ilorin, Nigeria

*Corresponding Author: Ghaniyyat Bolanle Balogun: balogun.gb@unilorin.edu.ng, folaadebisi2023@gmail.com

ARTICLE DATA

Article history:
Received 12 December 2023
Revised 26 June 2024
Accepted 24 July 2024
Available online

Keywords:
Intrusion Detection Systems (IDS), Machine Learning Models, Boruta Algorithm, PyCaret, Network Security

ABSTRACT

The paper discusses the development of intrusion detection systems (IDS) and their limitations in accurately detecting minority attack classes in computer networks. Despite advancements in IDS technologies, attackers can still breach networks. The aim of the work is to compare various machine learning models to find the best performing one for intrusion detection. The methodology involves using the Boruta algorithm for feature selection, under sampling to address class imbalance, and PyCaret for model comparison, training, and testing. The experimental results reveal that the Gradient Boosting classifier achieved the highest accuracy at 99.70%, while Naïve Bayes had the lowest accuracy at 84.77%. These findings underscore the importance of selecting robust machine learning approaches to enhance network security against evolving cyber threats. A stacking classifier was also created and outperformed other algorithms with 99.69% accuracy but slightly below the Gradient Boosting Classifier, which had 99.72% accuracy. The recommended model of choice for network intrusion detection is the Gradient Boosting classifier.

1. Introduction

The Internet has evolved into a critical tool in today's modern world. It helps people in all sorts of ways, such as business, recreation, and education [1]. It has been employed as a vital component of business models and operations, as is the case with every conglomerate, start-up, institution, or small-scale business [2].

From banking and credit reporting to space exploration, governance, education, and election security, computer networks are extremely crucial to the operation and functionality of computing systems, as well as to the survival of humanity as a whole. It is therefore imperative that network security be taken with high level of priority.

Computer networks are facing increased levels of threats, from hackers, to bots, worms, and other malicious software intended to cause great harm to organizations and institutions. The cyberspace has evolved a great deal from what it used to be years ago. As long as users are on the Internet or a computer network, be it wired, or wireless, cyber-thieves and hackers are looking for every means possible to exploit those system architectures responsible for providing the end users safety and security on such a network. It may be through monitoring of online presence, attempts to steal login details or impersonate users, hijacking their traffic, or converting networked computers into "an army of bots". Users/employees are any company or organization's biggest asset, but they are also their biggest weakness [3].

[4] remarked that a fully secure computerized system is "impossible to achieve" because as network security technology evolves, so do network exploitation technologies, hence the need for businesses,

information technology experts and firms to be alert and on the lookout for new attacks and react to them swiftly and appropriately.

[5] defines an intrusion as an action or step taken by someone which has some negative impact on the “confidentiality, integrity, or availability” of some information available on a network. [6] gave a more technical definition of an intrusion as any unauthorized access of a network or computer address in some domain. Intrusions can take different forms, from mild, crude ones like information gathering, to malicious, parasitic, data-extracting attacks like denial-of-service (DOS) attacks that can steal information and decimate large computer networks with security algorithms such as RSA. Intrusion detection renders assistance in detecting unauthorized network access or reduction in the quality of network performance (as well as network performance metrics) [7].

The idea of intrusion detection evolved from a novel idea vaguely proposed in Anderson’s 1980 paper on computer security [8] to full-fledged network security (intrusion detection and prevention) systems positioned at the frontlines of network defense. Intrusion detection can be manual or automated [9]. The main aim of intrusion detection is to continuously monitor a network in order to find any malicious activity or violation of network policy, and this is achieved through intrusion detection systems (IDS). One major difference between intrusion detection systems and other traditional network security measures, such as firewalls, is that an IDS detects a suspected intrusion once (and only once) it has taken place, and then notifies the network administrator/network security officer for further action, while a firewall basically restricts access to and between networks in order to prevent intrusion [10].

Distinguishing between normal and malicious network activities can be quite challenging to undergo manually, as it requires careful observation of, and inference from patterns in the logs of network traffic to find sequences of intrusion detection in network connections and transmissions [11].

[12] give two main types of intrusion detection, which are signature-based detection (also known as knowledge-based detection) and anomaly-based detection. In signature-based detection, previous records are utilized in order to find patterns similar to previous malware attacks. Anomaly-based detection, on the other hand, compares new network behaviour with normal network activities to look for any dissimilarities or abnormalities. If found, it flags it as an intrusion and reports to appropriate personnel, such as a system administrator or network engineer.

Signature-based detection may produce a lower false positive (or false alarm) rate, but can only detect known attacks, and requires frequent and periodic updates to the database to be effective [13]. Anomaly-based detection is more dynamic, but produces a higher false positive rate than signature-based detection, though it can be lower if a well-structured, relevant and reasonable dataset is utilized [14].

However, more than 30 years after the aforementioned report by Anderson [15], not much has changed in the area of intrusion detection, largely due to novel findings of network vulnerabilities, and difficulties obtained whilst coming up with a consensus as to “an accurate declaration of an intrusion”, and for many experts, intrusion detection had been viewed to be based mainly on “intuition and brute-force” [16]. As the internet has grown in popularity, attacks have become increasingly widespread. Penetrations, threats, hacks and all other forms of computer network attacks have become a continuously growing issue for computer network systems and their growth [17].

Network threats are activities not backed by the law and of malicious means. They are illegal or hostile operations that aim to exploit network vulnerabilities; thus, breach, break, steal or sabotage data and

information crucial to a person or an organization [18]. Network threats could be physical theft of data, abuse of power by an insider in an organization, malware attacks by outsiders, etc. [19].

[20] remarked that a fully secure computerized system is “impossible to achieve” because as network security technology evolves, so do network exploitation technologies. Therefore, a well-functioning network system must be up-to-date to stop threats, mitigate the damages and quickly recover from malicious attacks.

Network threats are growing robustly as technology grows [21]. Additionally, new hacking techniques are developed regularly to penetrate system communications and detect vulnerabilities of network systems. This shows that there is an ever increasing need to build networks and also improve already existing network systems to mitigate the damages caused by network threats and attacks.

Intrusion Detection System (IDS) and its importance cannot be overemphasised. In every organization that adopts the use of computer network systems to carry out operations, it is pertinent to deploy intrusion detection systems to be well-aware of potential malicious activities that could mar the smooth communication within the network system environment.

Any intrusion activity or violation found by an IDS is generally sent or forwarded to a network administrator, or gathered using a security information and event management (SIEM) system. A SIEM system integrates data from numerous points on the network and deploys filtering algorithms to distinguish between malicious and false alerts [22].

From the angle of method of deployment, network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS) are the two most popular IDS categories available. An HIDS observes important operating system files on a networked host computer, whereas a NIDS is a system that collates and analyses incoming flow of network traffic.

2. Methodology

2.1 Machine Learning

Machine learning is a component of artificial intelligence (AI) that gives machines the ability to “learn” through specialized algorithms that teach the machine to discover patterns and generate insights from the data such a machine is exposed to. For example, neural networks are used in different machine learning algorithms to mimic how the human brain would process data. Machine learning is sometimes referred to as predictive analytics in the business world. Machine learning algorithms create a model using training data in order to make predictions or choices without being explicitly taught or shown or programmed. They are used in a broad range of applications, including medical diagnosis, audio signal processing, speech recognition, detecting spam emails and computer vision, where developing traditional algorithms is very difficult or impossible.

2.2. Machine Learning Algorithms in Intrusion Detection Systems

With the shifting patterns of network behaviour, a dynamic method to detect and prevent such intrusions is required. It is widely agreed that detection based on static data do not reflect network traffic dynamisms, hence machine learning algorithms were used in enhancing network security [23]. Machine Learning techniques have recently been used in intrusion detection systems (IDS) to identify and categorize security threats. Some of the algorithms used include k-means algorithm, k-nearest neighbours, lazy algorithms, decision trees, random forests, rotation forests, support vector machines, Naïve Bayes classifier, neural networks, etc.

There have been quite a number of works exploring the importance of machine learning and the various methods employed towards detection of anomalies in computer networking. In recent years, artificial intelligence (and machine learning in particular) has grown to allow computers and devices to function

intelligently, in terms of data analysis and computing [24]. The massive growth and improvement in computing, network systems and technology in general has ushered in unwarranted threats, attacks and disturbances. There is an increasing need to build systems and improve existing computer networks to accurately detect intrusions, avoid attacks and rapidly correct and recover from attacks and anomalies. However, researchers over time have proposed, observed and analyzed methods that provide some understanding and progress towards intrusion detection in computer networks.

[25], one of the foremost researchers in this field, argued that in building a security monitoring surveillance system, one must first understand the risks and assaults that may be launched against a computer system, as well as how these threats can be mitigated. He examined the use of audit trails as a means of enhancing computer security. He outlined a consideration and created a general design of a system that proffers initial set of tools to computer system security users and officers to help in their work.

[26] stated that the first major intrusion detection was discussed by J.P. Anderson in 1980. They stated some techniques that have been used in previous years in intrusion detection. They include the Next-Generation Intrusion Detection Expert System (NIDES) created by SRI International in 1985; the Distributed Intrusion Detection System (DIDS), a joint effort by the United States Air Force, the University of California at Davis, the Haystack Laboratory, and the Lawrence Livermore Laboratory; the State

2.3.ProposedFramework

The research framework below shows the steps to be taken in carrying out this study.

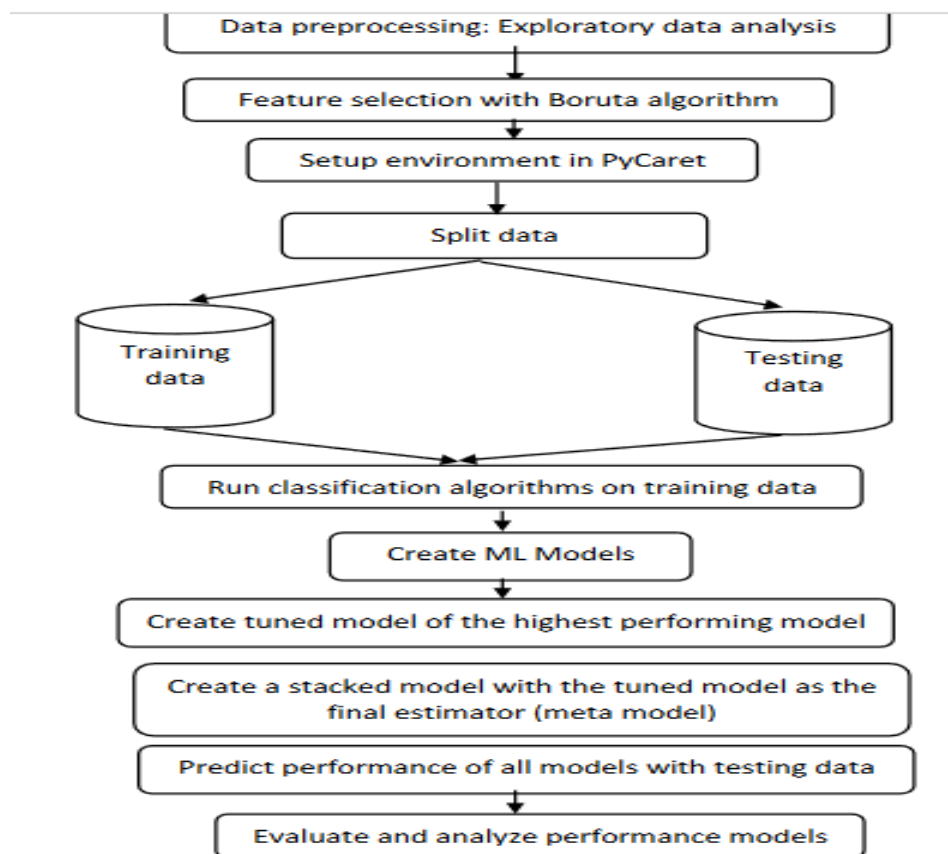


Figure 1: Proposed Framework

3.0 Method of Data Collection

The dataset used for this study is the CICIDS-2017 dataset (Canadian Institute for Cybersecurity Dataset), proposed by the Canadian Institute for Cybersecurity at the University of New Brunswick, Canada. This dataset was created to provide up-to-date common attacks, resembling actual network traffic data. [27] noted the shortcomings of popular datasets prior to 1998, which were often outdated and unreliable. Some datasets were skewed, lacked diversity, or abstracted packet data, failing to reflect current networking trends. Data capture spanned five days, split into morning and afternoon sessions, covering various traffic classes: Benign, Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack (XSS, SQL Injection), Infiltration, Botnet, and DDoS.

3.1 Feature Selection

Feature selection aims to reduce the feature space by eliminating non-useful features that could introduce noise and hinder model performance. The Boruta algorithm is utilized for this purpose.

Boruta Algorithm

The Boruta algorithm, introduced by M.B. Kursa and W.R. Rudnicki in 2010, simplifies machine learning experiments by identifying all important features that enhance model performance ([28]). Implemented as a wrapper around the random forest classifier, it differs from algorithms seeking a minimal-optimal feature set.

Setup Environment in PyCaret

Setting up the PyCaret environment involves importing the classification module (`pycaret.classification`). PyCaret determines data types for features, encodes categories, performs train-test splits, imputes missing values, and samples data when the sample size exceeds 25,000. The highest performing model is tuned using the Tree-structured Parzen Estimator for use as the meta model.

Predict Performance of Models

Models trained in PyCaret predict outcomes on test datasets, applying the transformation pipeline to prevent data leakage during experiments.

3.2. Programming Language

Python is selected as the programming language due to its simplicity, readability, versatility, and extensive libraries. It is widely used in web development, machine learning, and data research for both commercial and non-commercial purposes.

The dataset was obtained from the University of New Brunswick's Canadian Institute for Cybersecurity website at <https://www.unb.ca/cic/datasets/ids-2017.html>.

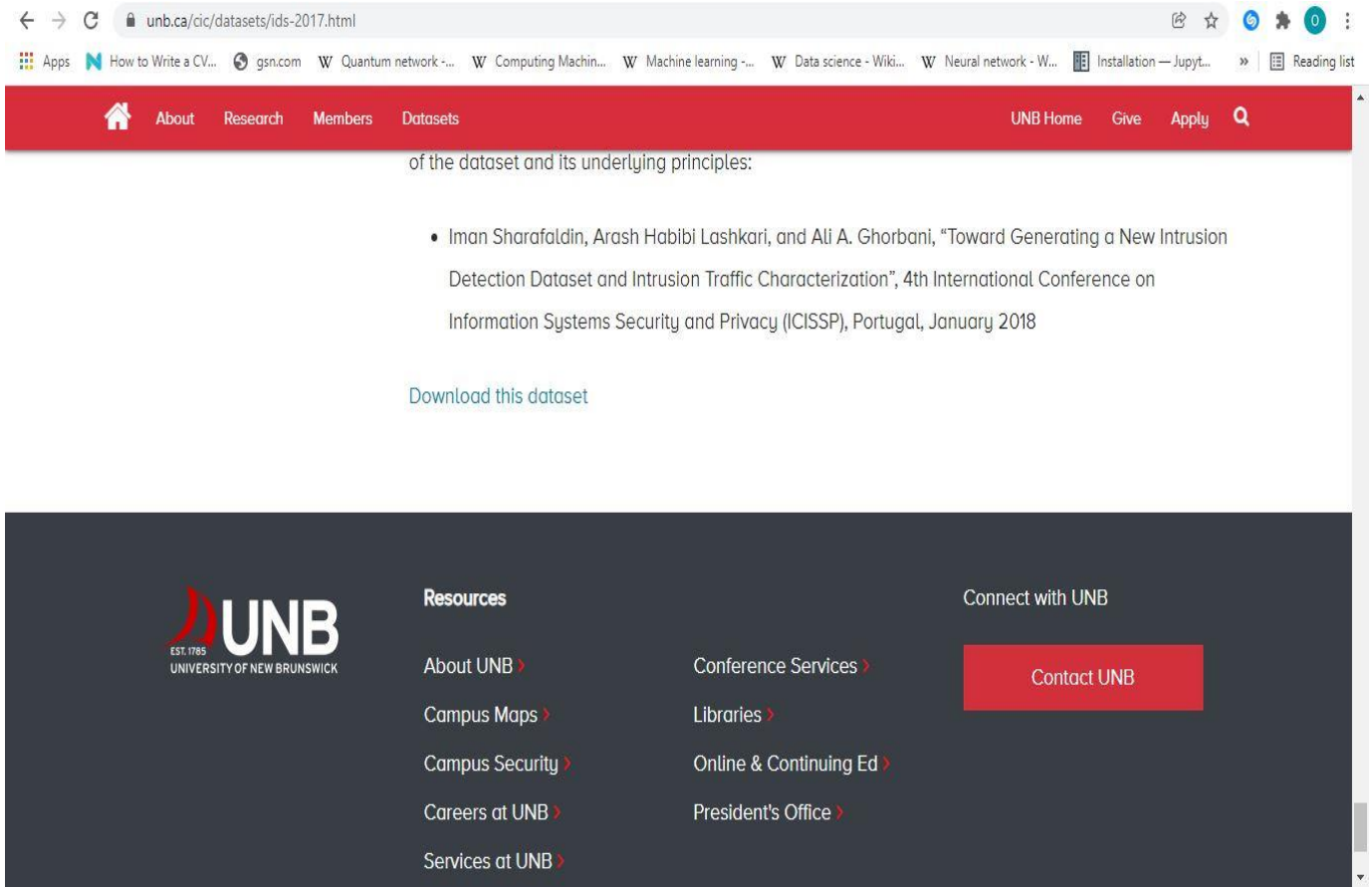


Figure 2 Downloading the dataset (to be deleted)

3.3. Downloading, Installing and Importing Required Libraries

Libraries such as PyCaret, along with packages like BorutaPy and Optuna (to be used for tuning), were downloaded and installed on the Google Colab environment. Relevant libraries were also imported for use in this notebook.

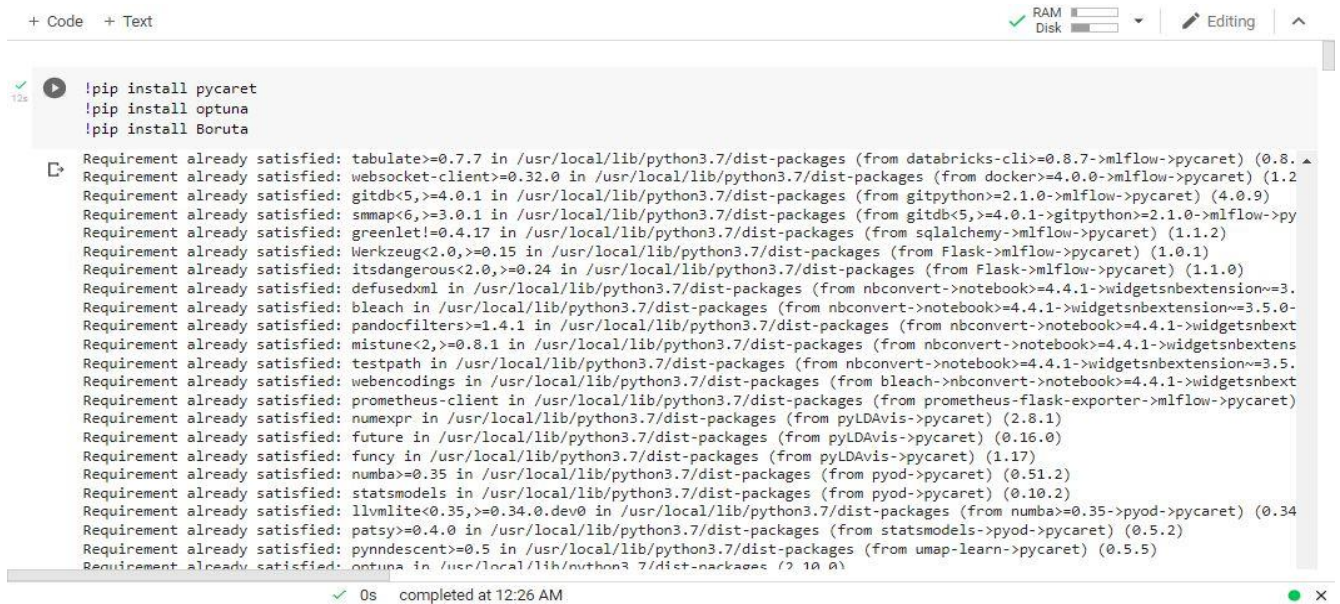


Figure 3 Installing packages

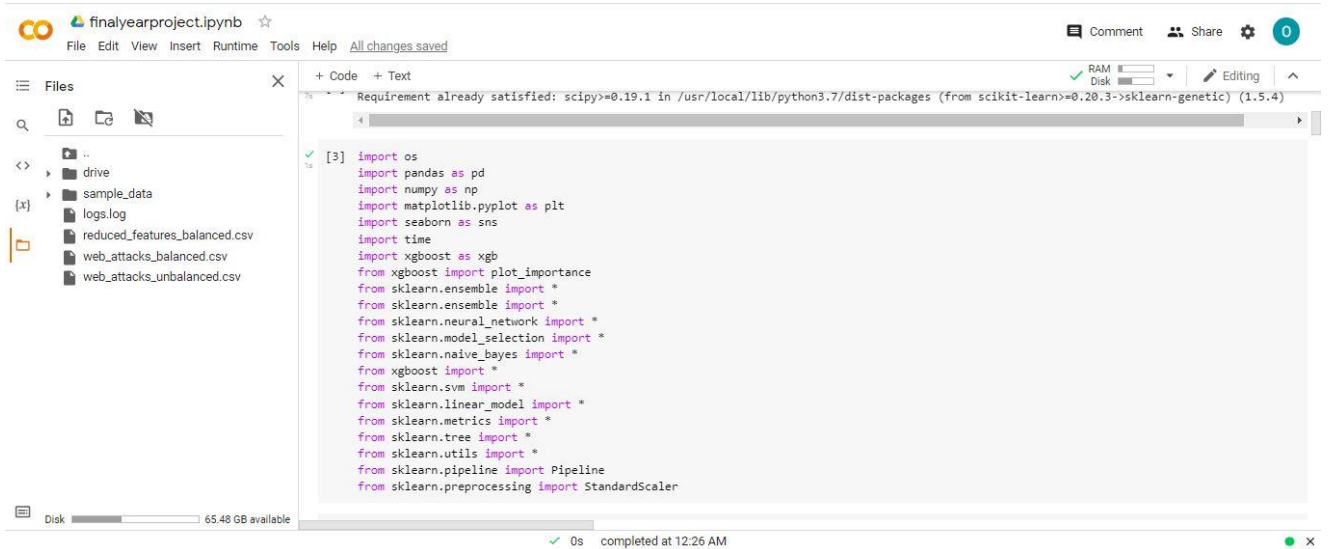


Figure 4 Importing relevant libraries for use

3.5. Data Exploration/ Pre-processing

One of the CSV files (Thursday Morning – Web Attacks) was selected for this study. The breakdown of the dataset is as follows:

Table 1: Breakdown of selected dataset file

Number of Instances	170366
Number of Features	79
Number of Classes	4 - Benign, XSS Attack, SQL Injection, Infiltration
Number of Numerical Features	52
Number of Categorical Features (excluding target variable)	26

Here, columns with empty/missing values were dropped. Also, feature selection using Boruta algorithm was performed, thus reducing the dataset’s dimensionality significantly. To achieve this, a Python package called BorutaPy was used. This package is based on the original R algorithm proposed in 2010, but with features such as direct interaction with Python libraries such as scikit-learn, faster run times, feature ranking and modularity (any ensembling technique could be used, such as random forest or even gradient boosting).

Additional features were dropped due to issues during preprocessing, and then undersampling was employed to significantly reduce the class imbalance in the dataset through the following technique:

1. The multi-class classification problem was reduced to a binary classification problem by collecting all the attack types into one single attack class and performing label encoding on the target variable.

2. The benign class was still significantly larger than the newly created attack class (with over 2000 instances), so an iterative undersampling approach was employed with the specific target of 70% benign records to 30% attack records. This reduced the benign attacks from over 168,000 to just slightly over 5000.
3. This was then saved into a new dataset CSV file to be used for our models

```

✓ [40] # Replace Inf values with NaN
    dataset = dataset.replace([np.inf, -np.inf], np.nan)
    # Drop all occurrences of NaN
    dataset = dataset.dropna()
    # Double check these are all gone
    dataset.isnull().any().any()

False

```

Figure 5 Dropping empty and infinity values from the dataset

```

✓ [16] # Replace Inf values with NaN
    dataset = dataset.replace([np.inf, -np.inf], np.nan)
    # Drop all occurrences of NaN
    dataset = dataset.dropna()
    # Double check these are all gone
    dataset.isnull().any().any()

#Feature Selection using Boruta algorithm
y = dataset[' Label']
columns = [' Label']
x = dataset.drop(columns = columns, errors='ignore')

from sklearn.ensemble import RandomForestClassifier

# define random forest classifier
forest = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
forest.fit(x, y)

RandomForestClassifier(class_weight='balanced', max_depth=5, n_jobs=-1)

```

FIGURE 6 Feature Selection using Boruta Algorithm – Initializing the classifier

4.0 Result


```

from boruta import BorutaPy

# define Boruta feature selection method
feat_selector = BorutaPy(forest, n_estimators='auto', verbose=2, random_state=1)

# find all relevant features
feat_selector.fit(x.values, y.values)

# check selected features
feat_selector.support_

# check ranking of features
feat_selector.ranking_

Iteration:    1 / 100
Confirmed:    0
Tentative:   76
Rejected:     0
Iteration:    2 / 100
Confirmed:    0
Tentative:   76
Rejected:     0
Iteration:    3 / 100

```

FIGURE 7: Implementing Boruta Algorithm using BorutaPy in 100 iterations

```

[24] # zip my names, ranks, and decisions in a single iterable
feature_ranks = list(zip(dataset.columns,
                        feat_selector.ranking_,
                        feat_selector.support_))

# iterate through and print out the results
for feat in feature_ranks:
    print('Feature: {:<25} Rank: {}, Keep: {}'.format(feat[0], feat[1], feat[2]))

```

Feature: Flow IAT Max	Rank: 1, Keep: True
Feature: Flow IAT Min	Rank: 1, Keep: True
Feature: Fwd IAT Total	Rank: 1, Keep: True
Feature: Fwd IAT Mean	Rank: 1, Keep: True
Feature: Fwd IAT Std	Rank: 1, Keep: True
Feature: Fwd IAT Max	Rank: 1, Keep: True
Feature: Fwd IAT Min	Rank: 1, Keep: True
Feature: Bwd IAT Total	Rank: 1, Keep: True
Feature: Bwd IAT Mean	Rank: 1, Keep: True
Feature: Bwd IAT Std	Rank: 2, Keep: False
Feature: Bwd IAT Max	Rank: 2, Keep: False
Feature: Bwd IAT Min	Rank: 1, Keep: True
Feature: Fwd PSH Flags	Rank: 10, Keep: False
Feature: Bwd PSH Flags	Rank: 19, Keep: False
Feature: Fwd URG Flags	Rank: 19, Keep: False
Feature: Bwd URG Flags	Rank: 19, Keep: False
Feature: Fwd Header Length	Rank: 1, Keep: True
Feature: Bwd Header Length	Rank: 1, Keep: True

0s completed at 12:26 AM

FIGURE 8: Showing the ranking of features based on their importance

```

▶ #Now we drop these features
excluded = ['Bwd IAT Std', 'Bwd IAT Max', 'Fwd PSH Flags', 'Bwd PSH Flags', 'Fwd URG Flags', 'Bwd URG Flags', 'FIN Flag Count', 'SYN Fla
dataset = dataset.drop(columns=excluded, errors='ignore')

[ ] dataset.shape

(170231, 49)

[ ] dataset['Label'].unique()

array(['BENIGN', 'Web Attack ⚡ Brute Force', 'Web Attack ⚡ XSS',
      'Web Attack ⚡ Sql Injection'], dtype=object)

[ ] dataset['Label'].value_counts()

BENIGN                168051
Web Attack ⚡ Brute Force    1507
Web Attack ⚡ XSS           652
Web Attack ⚡ Sql Injection    21
Name: Label, dtype: int64

```

FIGURE 9: Dropping rejected features

```

✓ [32] # We can either over-sample, or undersample. In this case, we undersample.
5s dataset.to_csv("web_attacks_unbalanced.csv", index=False)
dataset['Label'].value_counts()

BENIGN                168051
Web Attack ⚡ Brute Force    1507
Web Attack ⚡ XSS           652
Web Attack ⚡ Sql Injection    21
Name: Label, dtype: int64

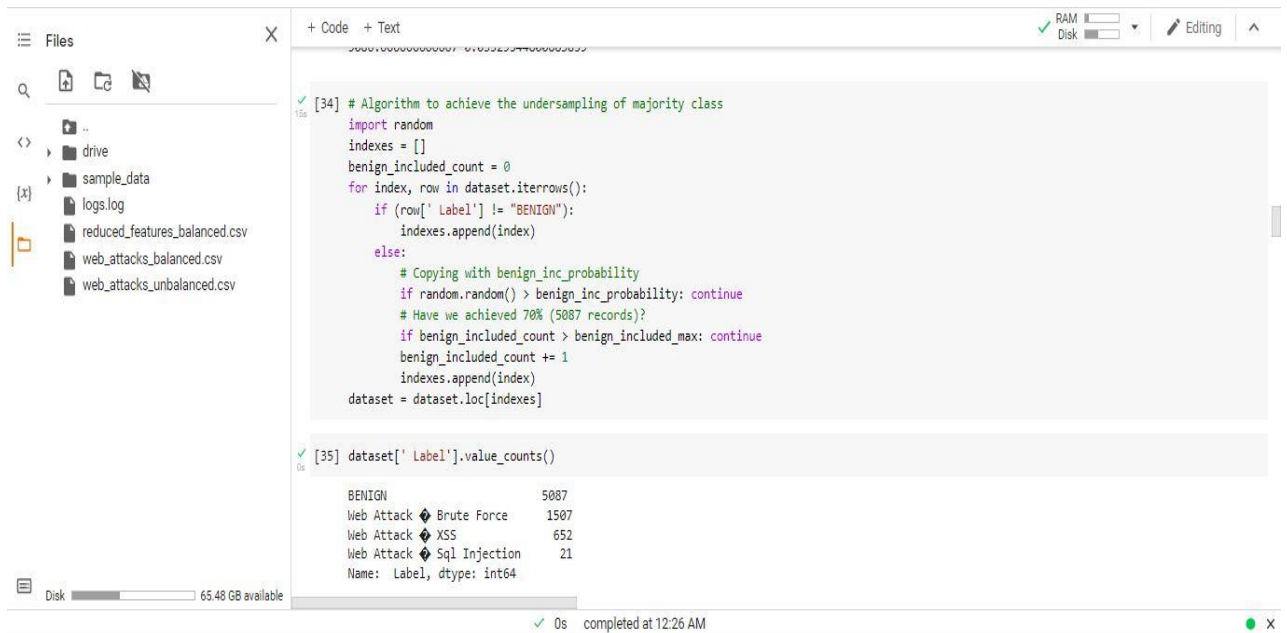
✓ [33] #All the records with the attacks are copied to the new dataset.
5s #Form a balanced dataset web_attacks_balanced.csv in proportion: 30% attack (2180 records), 70% benign data (2180 / 30 * 70 ~ = 5087 records)
#The enlargement multiplier is used to get exactly 70% benign data (5087 records).
enlargement = 1.1
benign_included_max = attack_total / 30 * 70
benign_inc_probability = (benign_included_max / benign_total) * enlargement
print(benign_included_max, benign_inc_probability)

5086.666666666667 0.03329544800883859

✓ 0s completed at 12:26 AM

```

FIGURE 10: Undersampling technique applied to the dataset



```
[34] # Algorithm to achieve the undersampling of majority class
import random
indexes = []
benign_included_count = 0
for index, row in dataset.iterrows():
    if (row[' Label'] != "BENIGN"):
        indexes.append(index)
    else:
        # Copying with benign_inc_probability
        if random.random() > benign_inc_probability: continue
        # Have we achieved 70% (5087 records)?
        if benign_included_count > benign_included_max: continue
        benign_included_count += 1
        indexes.append(index)
dataset = dataset.loc[indexes]

[35] dataset[' Label'].value_counts()

BENIGN          5087
Web Attack - Brute Force  1507
Web Attack - XSS         652
Web Attack - Sql Injection  21
Name: Label, dtype: int64
```

FIGURE 11: Code to perform the undersampling and the results

4.1. Setup Environment in PyCaret

The dataset is then split into training (60%) and testing (40%), with fold strategy set to StratifiedKFold with k = 10 as default, target variable identified and the MinMax scaler method for normalizing data. Here, PyCaret begins by inferring the correct data type for all features, and carrying out several minor tasks like categorical and one-hot encoding, train-test split, etc.

Once it brings out a prompt with all inferred data types, one must press Enter to confirm the inference or quit the setup as a whole, after which PyCaret will then show all the baseline parameters to be used throughout the machine learning experiment.



```
[45] #Setup environment

setup_project = pc.setup(data = dataset, target = ' Label', train_size=0.6,
                        fold_strategy = "stratifiedkfold",
                        normalize_method = 'minmax',
                        fold_shuffle = True,
                        use_gpu = True,
                        normalize = True,
                        preprocess = False,
                        html = True,

                        transformation = True,
                        verbose = True)
```

	Description	Value
0	session_id	8622
1	Target	Label
2	Target Type	Binary
3	Label Encoded	None
4	Original Data	(7267, 49)
5	Missing Values	False

FIGURE 12: Setting up the ML environment in PyCaret

4.2 Creating and Comparing ML Models

In this study, a total of nine algorithms were compared against each other during training. The metrics used were accuracy, precision, recall, F-1 score, Kappa statistic, MCC, and training times.

The results are shown below:

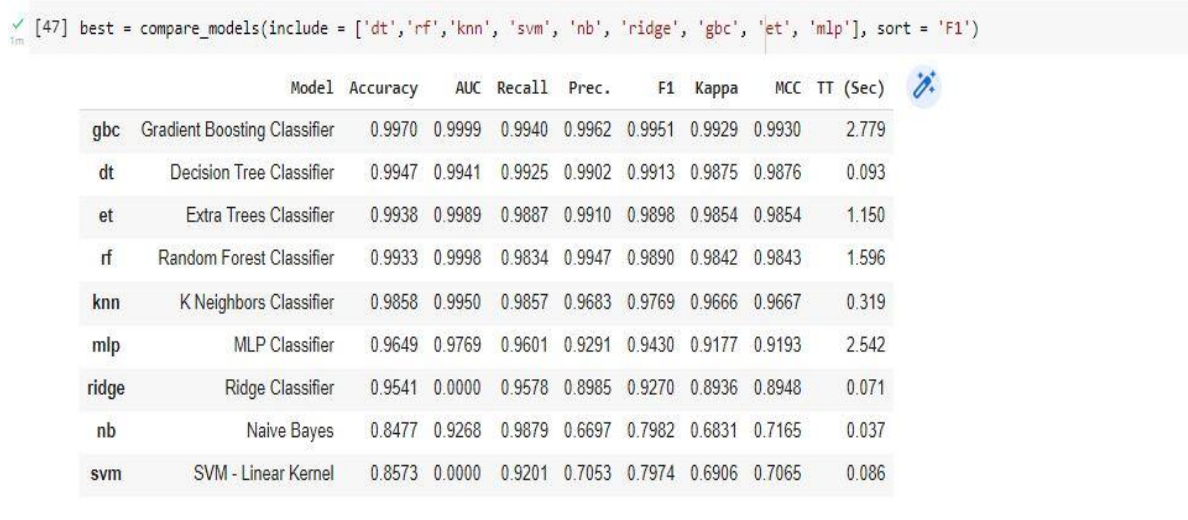


FIGURE 13: Comparing the performance of the models

Gradient Boosting Classifier had the highest accuracy with 99.70%, the longest training time (2.779 seconds), and outperformed all others in every other metric. However, the classifier with the fastest training time was Naïve Bayes, with a time of 0.036 seconds. Five of the best models (Gradient Boosting, MLP, Decision Tree, Random Forest, and Extra Trees) were selected and then created. When creating a model, the model is run on the same metrics with a k-fold of 10 by default, meaning the model is run 10 times, and the mean of all 10 runs is taken as the metric. If there is a high standard deviation between the runs, it tells us that the model may not be as effective as it might appear to be.

4.8. Hyperparameter Tuning of Best Classifier

The highest performing classifier was then tuned using a Tree-structured Parzen estimator (TPE), an optimization algorithm (that utilizes Gaussian Mixture Models (GMM)) under the Optuna package. This was done to create a trained ML model to be used as the final meta model/estimator for the ensemble classifier.

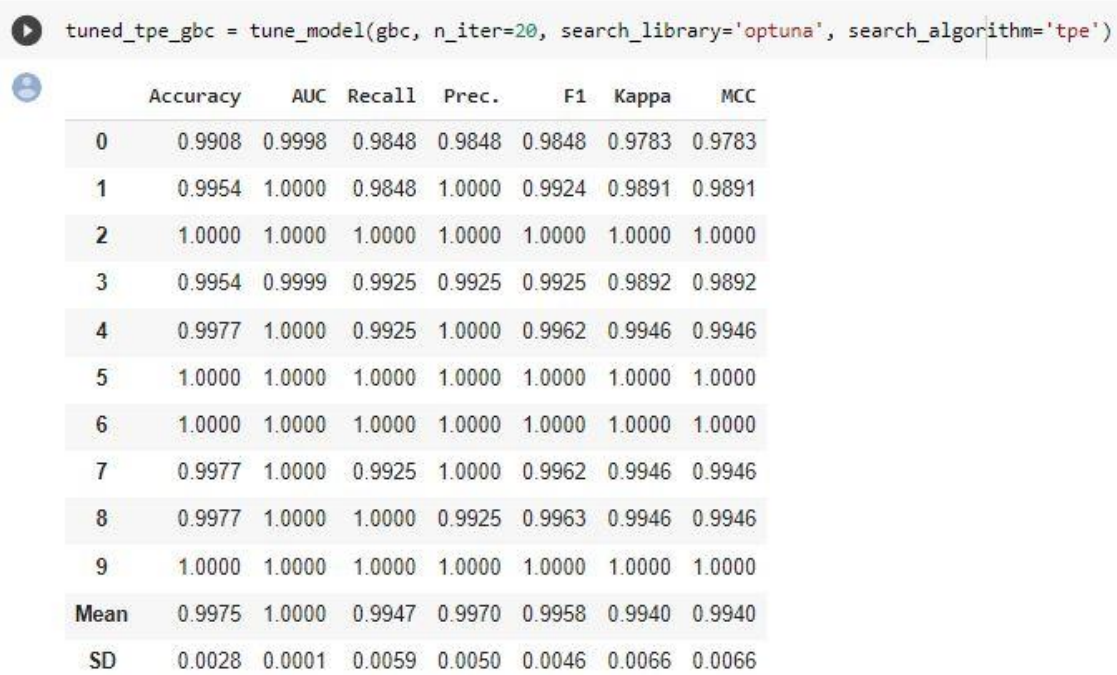


Figure 14: Tuned Gradient Boosting Classifier running on 20 iterations

4.3. Hybridization/Stacking

This is the process by which several base learners/classifiers are hybridized, combined, or “stacked” on each other – in such a way that each learner layer feeds the next learner until a final output is provided by the estimator model or meta model. It is an ensembling technique.

The structure of the stacked classifier model is shown below:

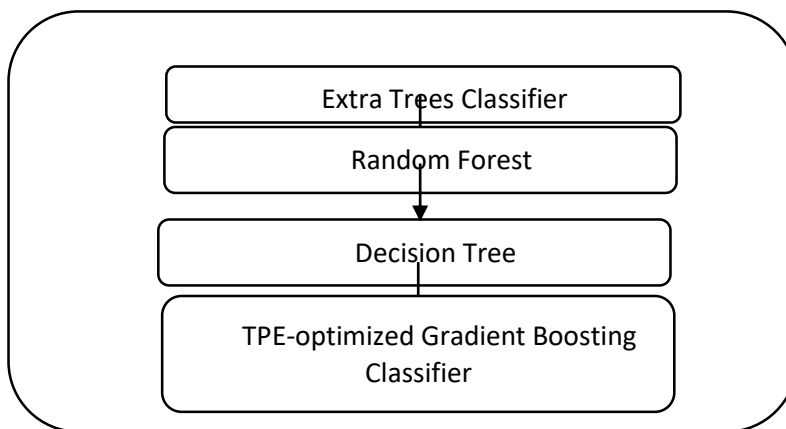


FIGURE 14: Structure of proposed StackingClassifier

The Stacking Classifier had an accuracy of 99.84%, higher than that of Gradient Boosting Classifier.

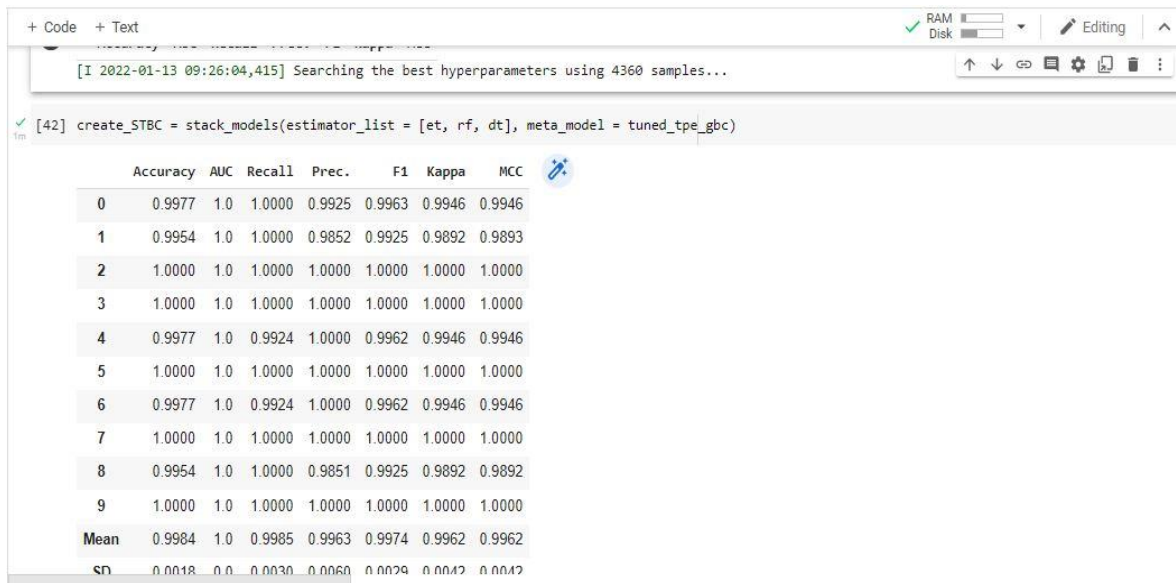


FIGURE 15 Predicting model performance of Stacking Classifier

4.4. Testing /prediction of models

The Stacking Classifier was then compared against Gradient Boosting Classifier, Decision Tree, Random Forest, and Multi-layer Perceptron, using the testing dataset. It outperformed all others with an accuracy of 99.69% against 99.52% from Decision Tree, 99.59% from random Forest, 99.55% from Extra Trees Classifier, and 97.45% from multi-layer perceptron. It however, performed slightly lower than Gradient Boosting Classifier, which had 99.72% accuracy.

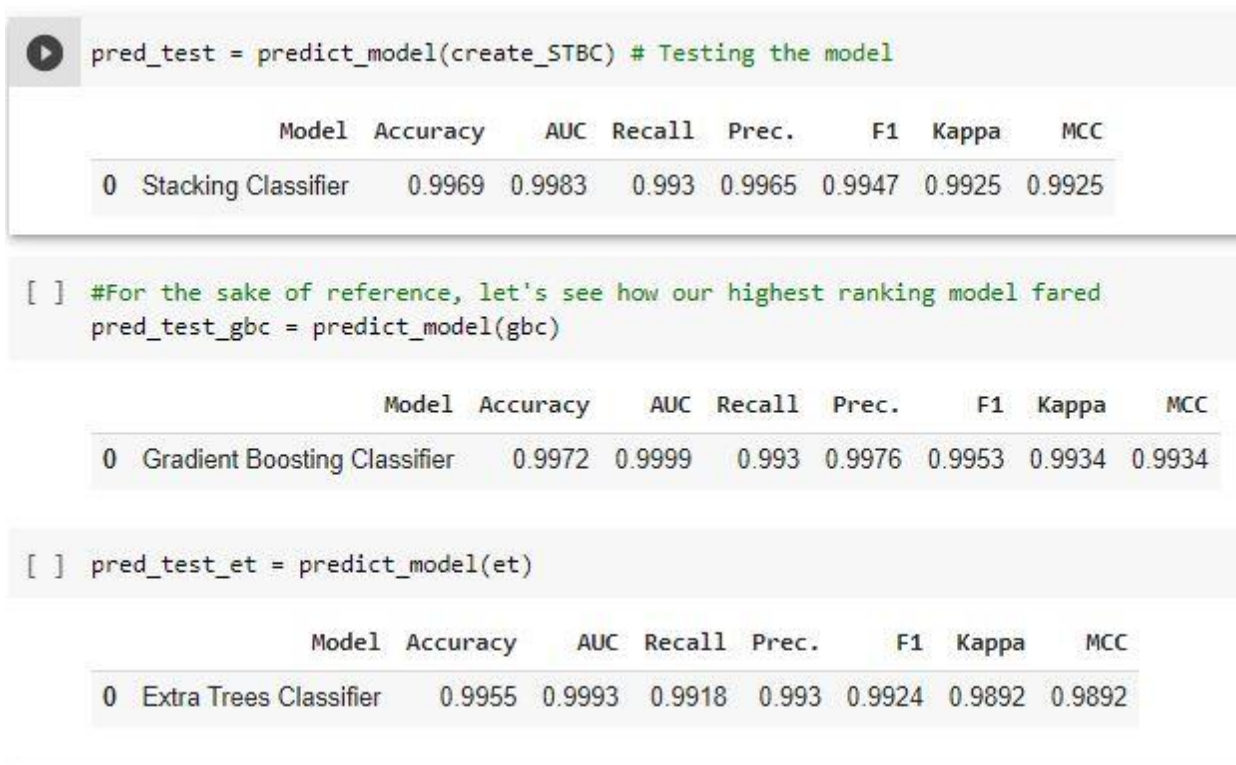


FIGURE 16: Predicting model performance on testing data

4.5. Evaluation and Interpretation of Performance Metrics

The classification reports and confusion matrices for the models that underwent testing are shown below:

(a.) Decision Tree:

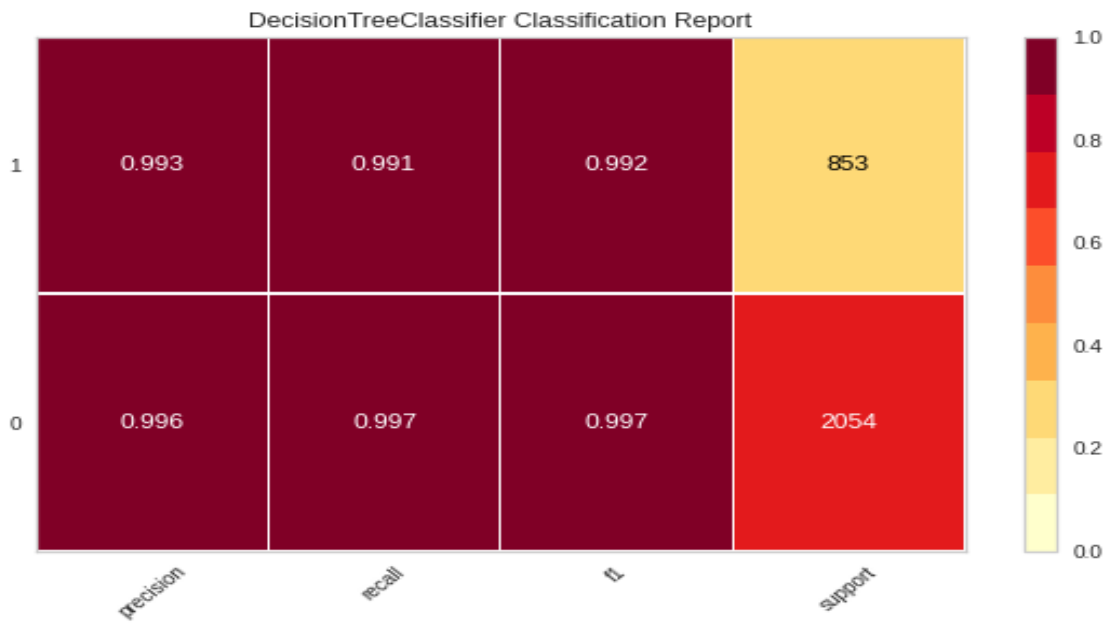


FIGURE 17: Decision Tree Classification Report

This shows very good metrics from the minority class (largely due to undersampling and feature selection), showing that the issue of class imbalance has been improved significantly.

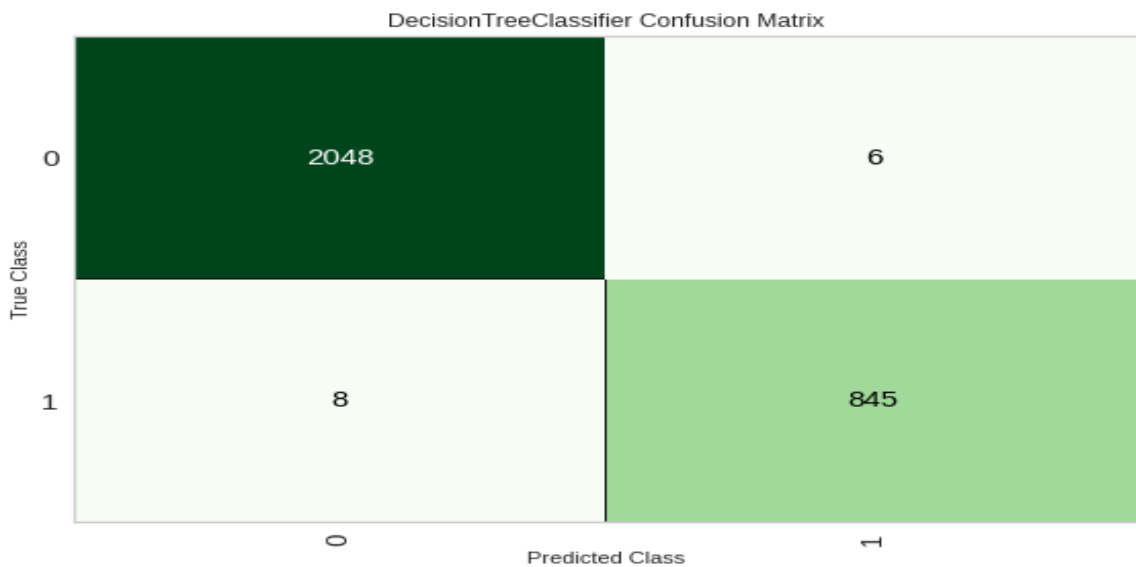


FIGURE 18: Decision Tree Confusion Matrix

The number of false positives is quite low, just 8 instances. The False Positive Rate (FPR) for this classifier would be determined as:

$$FPR = \frac{FP}{TP+FN} = 8/(2048+6) = 8/2054 = 0.0039$$

This is quite low, a good sign that the model is reliable.

(b.) Random Forest: This model has a weaker minority recall than Decision Tree, but a higher minority precision.

It is also noticed that the precision for the minority class is higher (99.8%) than that of the majority class (99.5%).

From the confusion matrix, the FPR would be: $10/(2052+2) = 10/2054 = 0.0049$, slightly higher than that of Decision Tree.

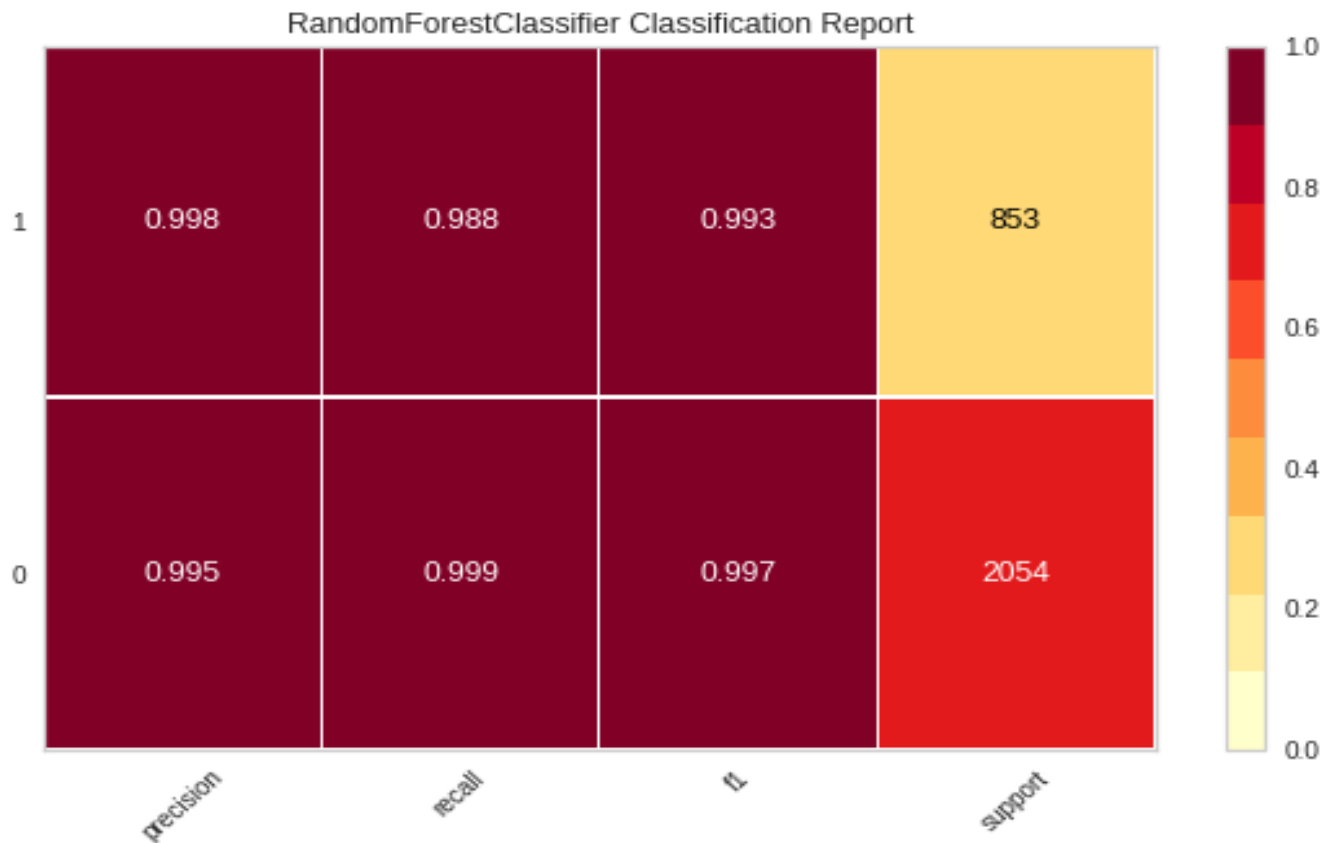


FIGURE 19: Random Forest Classification Report

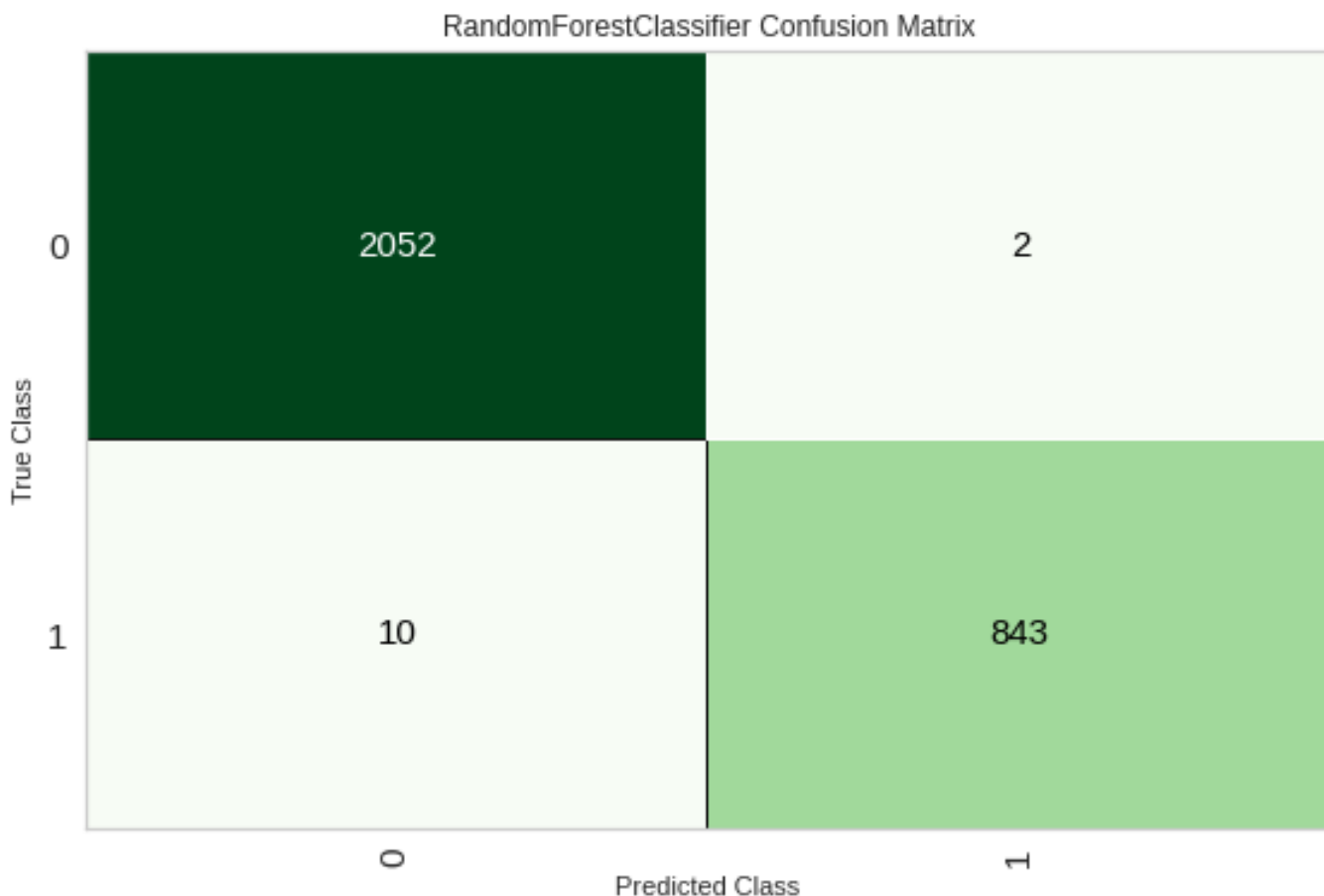


FIGURE 20: Random Forest Confusion Matrix

(c.) Multi-Layer Perceptron: This classifier has the lowest minority class performance of all models that went through testing phase. However, recall score for minority attack class is higher (98.2%) than that of majority benign class (97.1%).

The FPR of this model, as obtained from the confusion matrix is: $15/(1995+59) = 0.0073$, higher than any other model here.

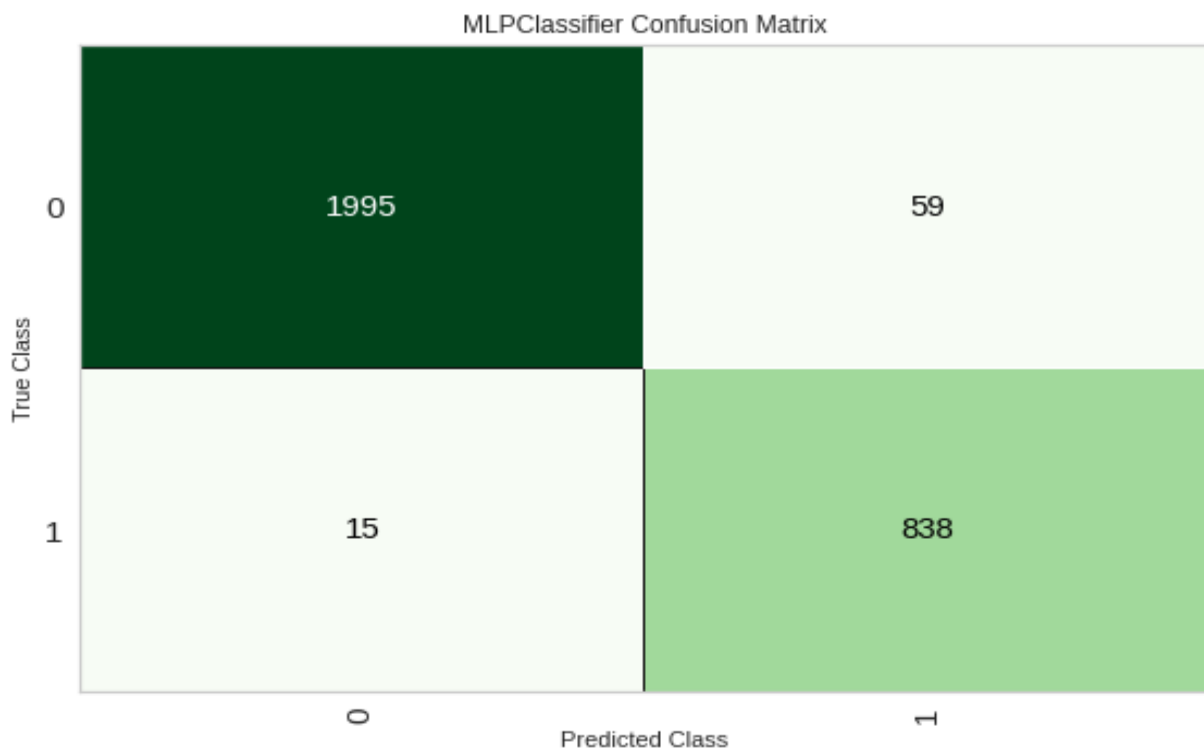


FIGURE 21: MLP Classifier Confusion Matrix

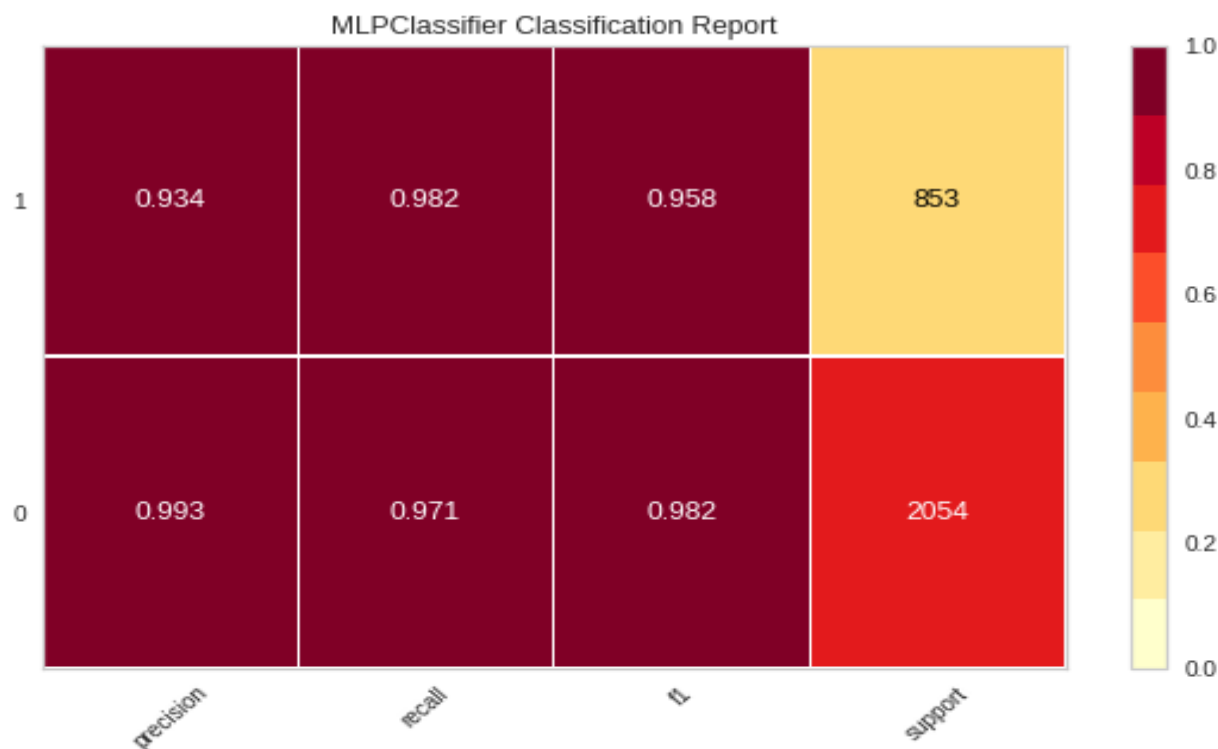


FIGURE 22: MLP Classifier Classification Report

(d.) Extra Trees Classifier: This classifier has a lower number of false positives that Decision Tree Classifier. The classifier’s FPR is determined as: $FPR = 7/(2048+6) = 0.0034$, the lowest in the set.

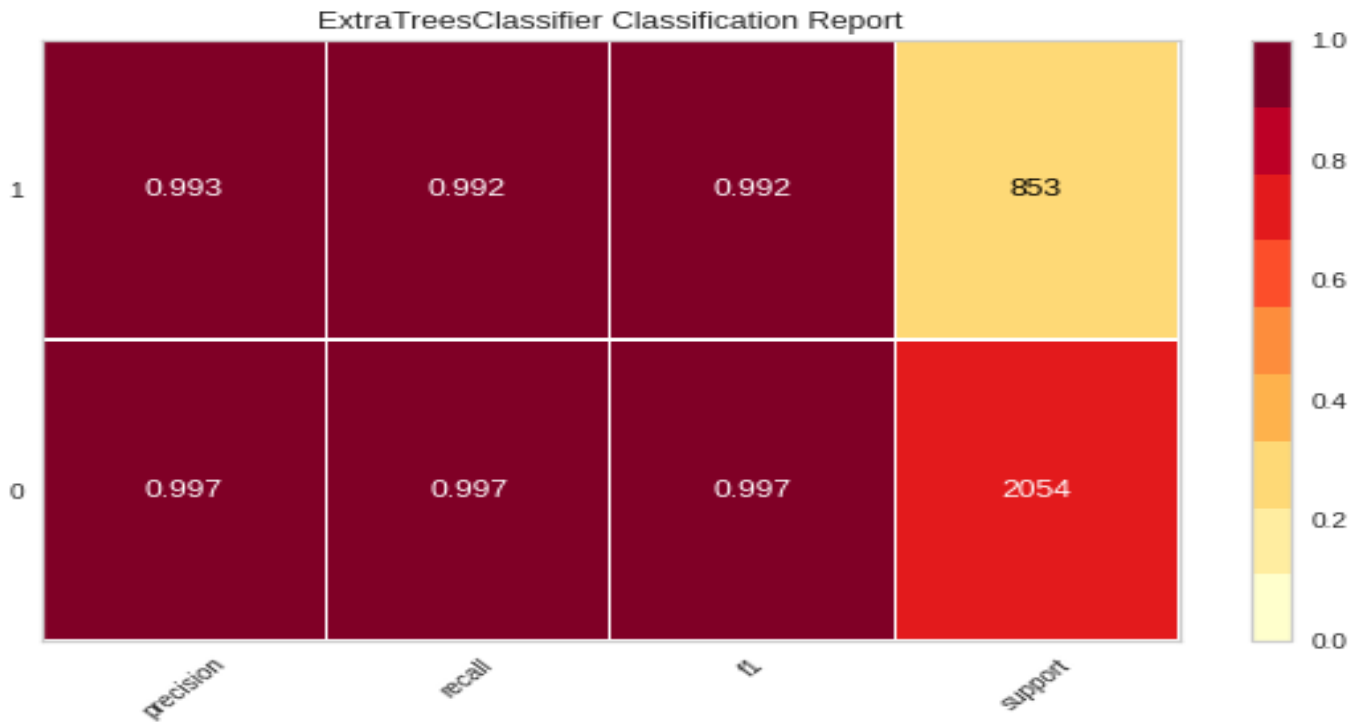


FIGURE 23: Extra Trees Classifier Classification Report



FIGURE.24: Extra Trees Classifier Confusion Matrix

(e.) Gradient Boosting Classifier: This classifier performed quite well, with a majority class precision of 99.6%, recall of 99.8% (second only to Random Forest) and f-1 score of 0.997. The FPR score is determined as $FPR = \frac{8}{2047 + 4} = 0.0039$, same as the Decision Tree Classifier.

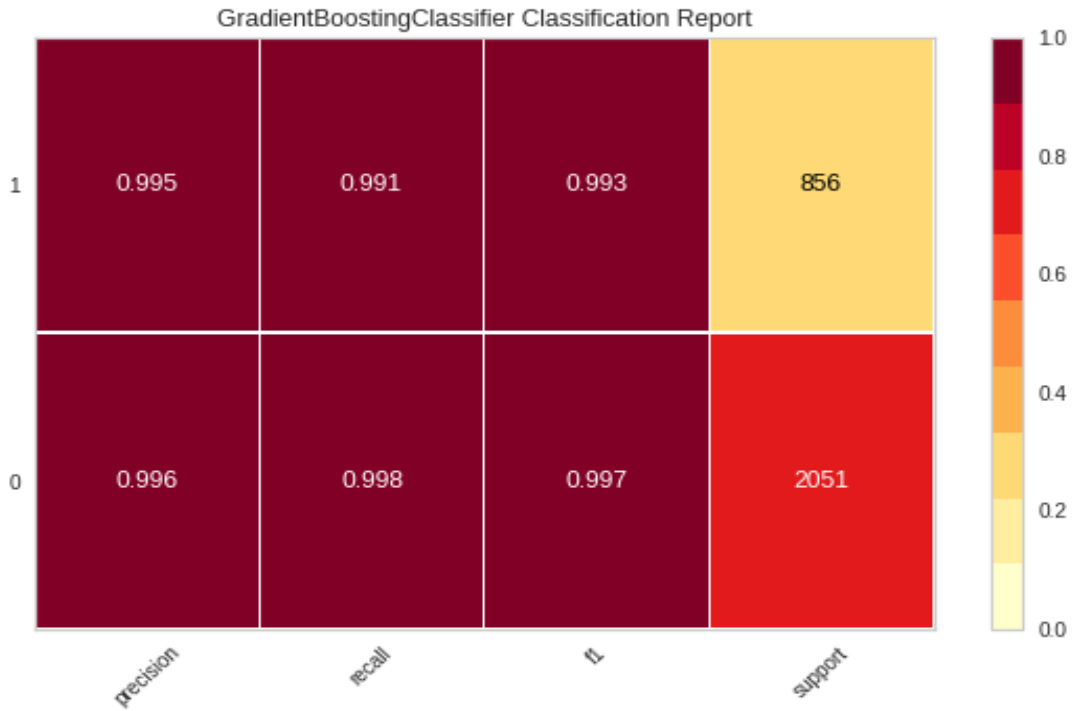


Figure 25: Gradient Boosting Classifier Classification Report

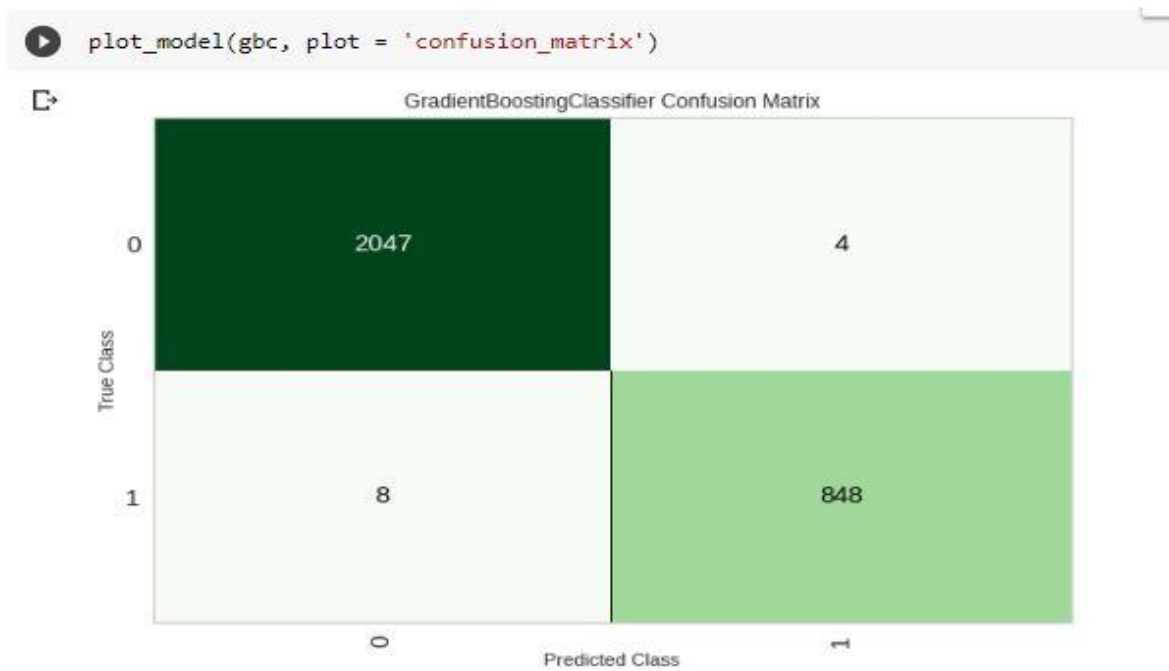


Figure 26: Gradient Boosting Classifier Confusion Matrix

(f.) Stacking Classifier: This classifier has very impressive majority and minority class metrics, and an FPR of $12/(2054 + 0) = 0.0058$.

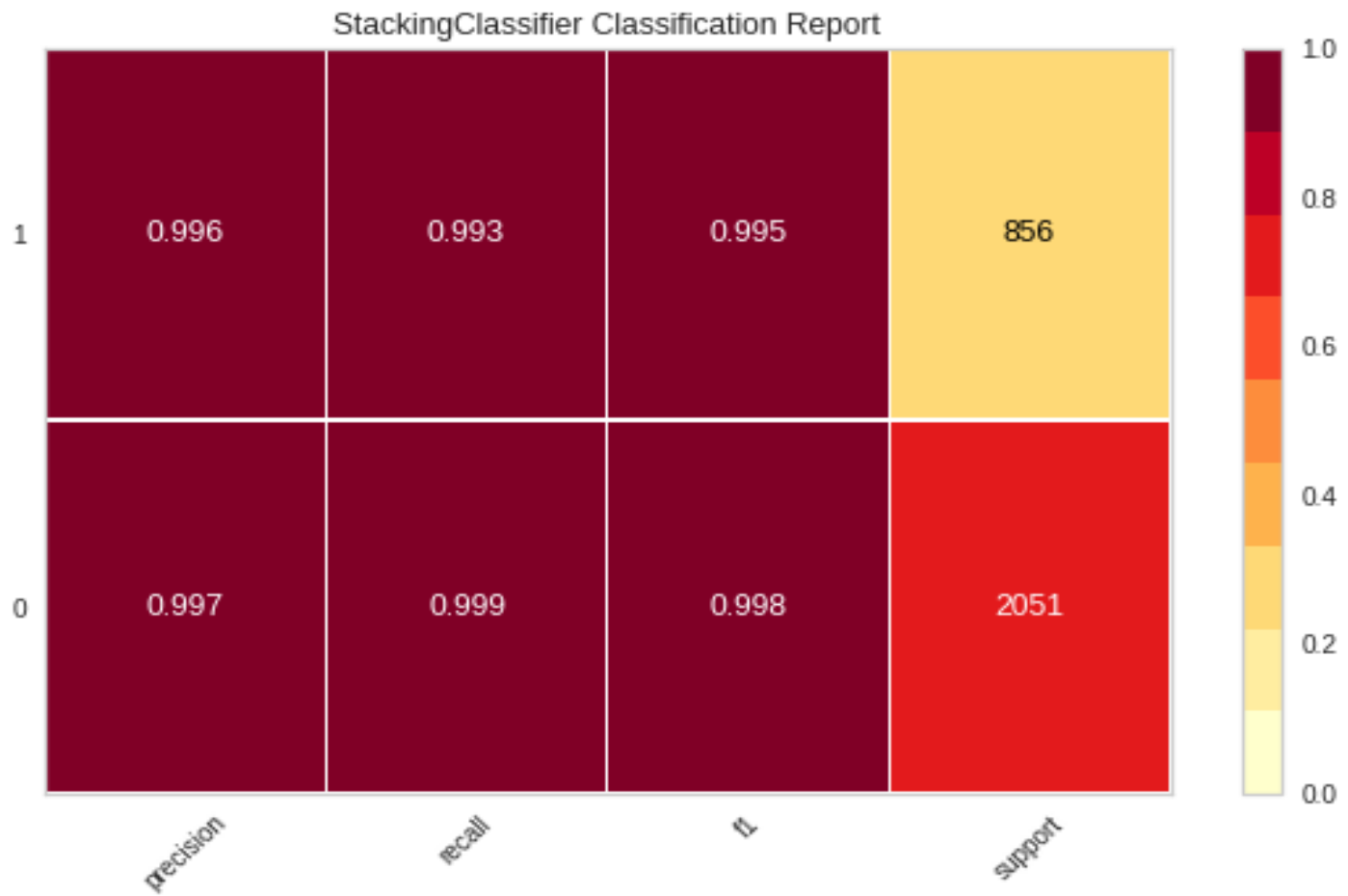


Figure 27: Stacking Classifier Classification Report

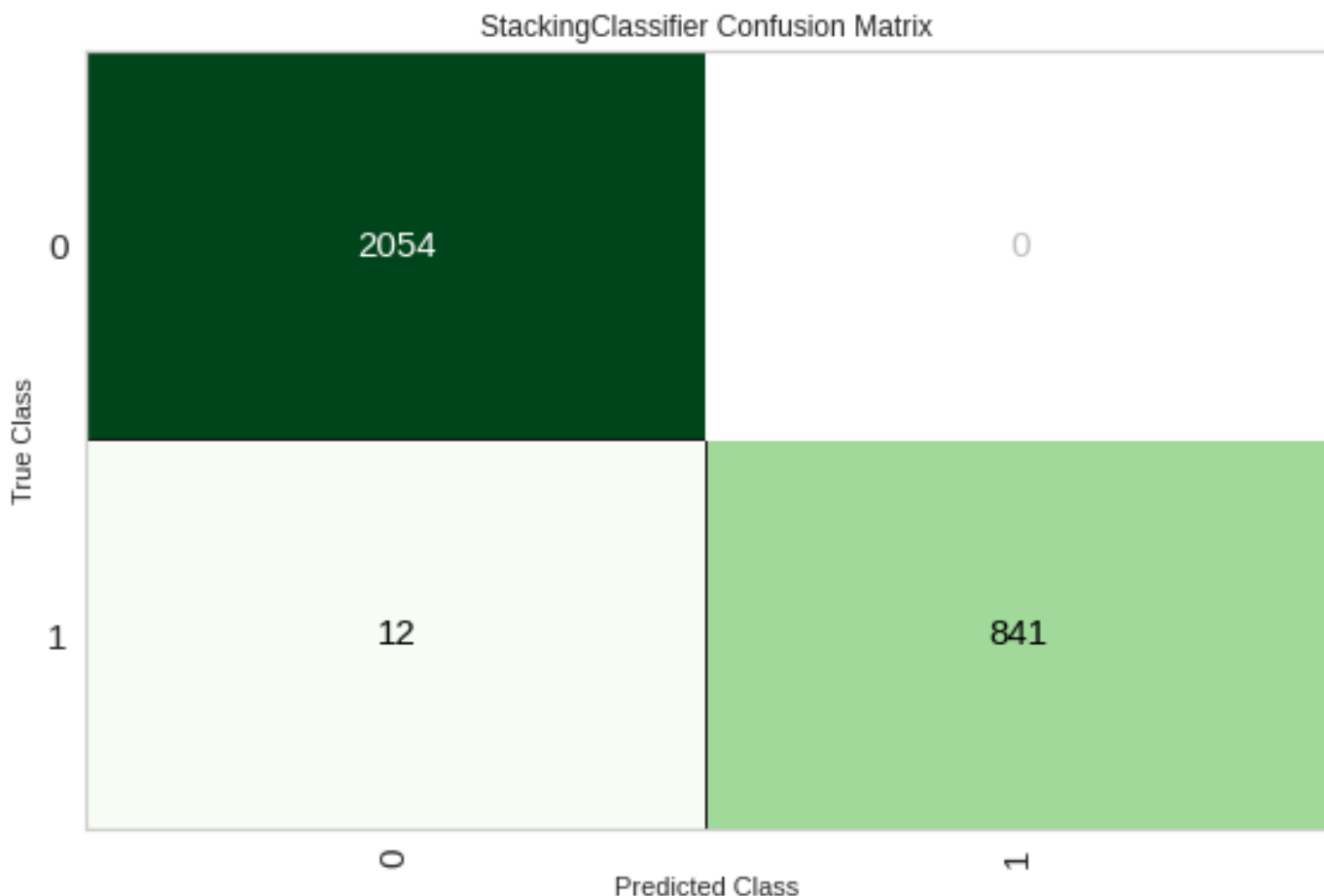


FIGURE 28: stacking classifier confusion matrix

5.0 Conclusion

Machine learning techniques help solve the problem of intrusion detection by classifying the condition based on the logs of network traffic gotten from a given dataset. However, many datasets are either largely skewed towards a particular attack type, not considering many newer forms of network attacks, not covering a wide range of known attacks, or abstracting packet data, which cannot reflect modern networking trends and practices.

The implementation employed in this study made use of Boruta algorithm for feature selection, undersampling to tackle class imbalance, PyCaret to compare, train and test models, including creating the stacking classifier model, and obtain visual plots to interpret the performance of these models.

The experimental results obtained show that on a dataset of over 7000 network data instances, Gradient Boosting classifier had the highest accuracy with 99.70% accuracy, while Naïve Bayes had the lowest accuracy with 84.77%. Linear SVM had the lowest F-1 score with 0.7974, while Naïve Bayes had the fastest training time with 0.037 seconds and Gradient Boosting classifier had the longest training time in 2.779 seconds.

A stacking classifier was created with a TPE-optimized Gradient Boosting classifier as the final estimator/meta model and its performance compared against 5 selected algorithms – Decision Tree, Random Forest, Extra Trees, Gradient Boosting, and Multi-layer Perceptron. It outperformed all others with an accuracy of 99.69% but performed slightly lower than Gradient Boosting Classifier, which had 99.72% accuracy.

In terms of false positive rate (FPR), Extra Trees had the lowest FPR with 0.0034, followed by both Decision Tree and Gradient Boosting with an FPR of 0.0039, while MLP had an FPR of 0.0073.

This shows that the methodology used in this study is effective in classifying network data into normal and malicious traffic, and should be used when developed machine learning-based intrusion detection systems.

References

- [1]. Anderson, J.P. (1980). Computer Security threat monitoring and surveillance Report, James P. Anderson Company. Washington: James P. Anderson Co.
- [2]. West, M. (2014). Network and System Security (2nd ed.). Boston: Syngress.
- [3]. Verma, A., & Ranga, V. (2018). Statistical analysis of CIDDS-001 dataset for network intrusion detection systems using distance-based machine learning. *Procedia Computer Science*, 125, 709-716.
- [4]. Solane, D., & Omar, D. N. (2015). Using Data Mining Algorithms for Developing a Model for Intrusion Detection System (IDS). *ELSEVIER*.
- [5]. Gharib, A., Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2016, December). An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)* (pp. 1-6). IEEE.
- [6]. Kursu, M. B., & Rudnicki, W. R. (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11), 1–13. <https://doi.org/10.18637/jss.v036.i11>
- [7]. Chellam, A., Ramanathan, L., & Ramani, S. (2018). Intrusion detection in computer networks using lazy learning algorithm. *Procedia computer science*, 132, 928-936.
- [8]. Sarker, I. H. (2021). Ai-driven cybersecurity: an overview, security intelligence modeling and research directions. *SN Computer Science*.
- [9]. Budler, B., & Ajoodha, R. (2021). Comparative Analysis of Deep Learning Models for Network Intrusion Detection Systems, pp. 1-4.
- [10]. Cannady, J., & Harrell, J. (1996). A comparative analysis of current intrusion detection technologies. In *Proceedings of the Fourth Technology for Information Security Conference (Vol. 96)*.
- [11]. Ahmad, I., Abdullah, A. B., & Alghamdi, A. S. (2010). Applying neural network to U2R attacks. *2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA)*. <https://doi.org/10.1109/ISIEA.2010.5679451>
- [12]. Man, J., & Sun, G. (2021). A Residual Learning-Based Network Intrusion Detection System. *Security and Communication Networks*, 2021.
- [13]. Richa, P., & Adesh, N. D. (2021). Comparative analysis of Machine Learning algorithms for Intrusion Detection. In *IOP Conference Series: Materials Science and Engineering (Vol. 1013, No. 1, p. 012038)*. IOP Publishing.
- [14]. Chakraverty, P. M. (2015). Computer Networking Technologies and Application to IT Enabled Services.
- [15]. Osisanwo, F. Y., Akinsola, J. E. T., Awodele, O., Hinmikaiye, J. O., Olakanmi, O., & Akinjobi, J. (2017). Supervised Machine Learning Algorithms: Classification and Comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3), 128-138. doi: 10.14445/22312803/IJCTT-V48P126
- [16]. Gaigole, M. S., & Kalyankar, M. A. (2015). The Study of Network Security with Its Penetrating Attacks and Possible Security Mechanisms. *International Journal of Computer Science and Mobile Computing*, 4(5).
- [17]. Delamore, B., & Ko, R. K. L. (2015). Security as a service (SecaaS)—An overview. *The Cloud Security Ecosystem*, 187–203. doi:10.1016/b978-0-12-801595-7.00009-4
- [18]. Lin, S.-W., Ying, K.-C., Lee, C.-Y., & Lee, Z.-J. (2012). An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection. *Applied Soft Computing*, 12(10), 3285–3290. doi:10.1016/j.asoc.2012.05.004
- [19]. McHugh, J., (2001). Intrusion and Intrusion Detection. *IJIS*, 2001(1). pp. 14–35. doi:10.1007/s102070100001. Pittsburgh: Springer-Verlag.
- [20]. Ko, R., & Choo, R. (2015). The cloud security ecosystem: technical, legal, business and management issues. Syngress.
- [21]. Mighan, S. N., & Kahani, M. (2021). A novel scalable intrusion detection system based on deep learning. *International Journal of Information Security*, 20(3), 387-403. doi:10.1007/s10207-020-00508-5

- [22]. Sheikh, T., Syed, R., Rayan, A., & Saleh, A. (2019). An adaptive intrusion detection and prevention system for Internet of Things. *International Journal of Distributed Sensor Networks*, 15(11).
- [23]. Kumar, D. B., & Deepa, B. (2015). Computer Networking: A Survey. *International Journal of Trend in Research and Development*, 2(5).
- [24]. Gaylord, I. (2021). Network Intrusion: How to Detect and Prevent It. Retrieved from United States Cybersecurity Magazine: <https://www.uscybersecurity.net/network-intrusion/>
- [25]. Tsai, C. F., Hsu, Y. F., Lin, C. Y., & Lin, W. Y. (2009). Intrusion detection by machine learning: A review. *Expert systems with applications*, 36(10), 11994-12000.
- [26]. Gaylord, I. (2021). Network Intrusion: How to Detect and Prevent It. Retrieved from United States Cybersecurity Magazine: <https://www.uscybersecurity.net/network-intrusion/>
- [27]. West, M. (2014). Preventing system intrusions. In *Network and System Security* (pp. 29-56). Syngress.
- [28]. Kursu, M. B., & Rudnicki, W. R. (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11), 1–13. <https://doi.org/10.18637/jss.v036.i11>