INTRUSION DETECTION SYSTEM

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and

Sciences

Submitted by

Kosana Raghu Sai (AP20110010311)

Tangirala Tarun (AP20110010329)

Kodali Sai SriLatha (AP20110010713)

Gajula Siva Sai (AP20110010335)

Under the Guidance of

Dr. Hemantha Kumar Kalluri

SRM University-AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh - 522 240

[May 2023]

Certificate

Date: 13-Dec-22

This is to certify that the work present in this Project entitled "**Intrusion Detection**" has been carried out by **Raghu Sai** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in the **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. / Dr. [Name]

Designation, Affiliation.

Co-supervisor

(Signature)

Prof. / Dr. [Name]

Designation, Affiliation.

Acknowledgments

I would like to express my gratitude to our mentor, Dr. Hemantha Kumar Kalluri who guided us through our project. I am thankful to our mentor for providing such a wonderful opportunity and introducing us to so many new things.

I would also like to thank my team members who has contributed in this project and helped me to complete this project.

Table of Contents

Abstract

Cyberattacks are increasing quickly, and the security services have suffered greatly as a result of their inability to stop the infiltration. Network intrusion detection systems (NIDS) are frequently used to check for unauthorized network access that is obtained by malicious actions. With the help of algorithms like KNN, Naive Bayes (Gaussian) Algorithm, XgBoost Classifier, RandomForestTree Classifier, Decision Tree Classifier, and others, this project suggests a method for effectively detecting intrusions. The experiments will be conducted in a TON-IoT Data set, although they will be more heavily focused on Train-Test Data sets in order to configure the effectiveness of the algorithms.

Based on accuracy, recall, and f1score, our proposed approach is expected to produce the best results for Binary Class and Multi Class when compared to the earlier proposed algorithms.

1. Introduction

Nowadays, metropolitan cities are seeking to combine Internet of Things, blockchain technologies, and edge computing to improve the capabilities of their operations and provide the finest services to their citizens. Security and privacy are the two issues with IOT systems that make it difficult to supply these services. One of the technologies that solves these concerns is intrusion detection.

The process of monitoring system traffic, analyzing it, and spotting abnormal activity is known as **Intrusion detection.** The intrusion detection system uses two different detection techniques. One is the signature-based method, which identifies suspicious activity based on specific patterns, such as the number of 0s or 1s, and so forth. Based on known malicious activities, it can identify suspicious activity. Signatures are the patterns that are recognized. This signature-based intrusion detection system is able to identify suspicious activity with known signatures, but it has difficulty identifying suspicious activity with unknown signatures or patterns.

The second strategy called an Anomaly-based approach, finds suspicious activity that is produced quickly and whose signature is unknown in advance. This anomaly-based system evaluates the suspicious actions with the machine learning models it utilizes. If they are absent from that model, it is deemed an attack. This anomaly-based IDS is superior to signature-based IDS in many ways.

1.1 Available Datasets

Now that deep learning and machine learning models are being used for intrusion detection, huge datasets are needed to determine the validity and consistency of those models.

Among the accessible datasets are the KDD99 and NSL-KDD data-sets, and CAIDA data sets, Among these are the DEFCON data set, the UNIBS data set, the LBNL data set, the Kyoto data set, the Darpa2009 data set, the Bot-IOT data set, the UNSW-NB15 data set, the CICIDS2017 data set, and so on.

The key drawbacks of the datasets include that they are obsolete, the absence of actual network traffic, missing values, a high degree of class imbalance, and the fact that the data is stored in files. The most recent and most recent IOT assaults are not included. It is not viable to evaluate the new AI models using some of the aforementioned data sets because they only contain attack events and no routine occurrences. Due to the fact that each data set was created to validate only a single cybersecurity solution, these data sets could not be used to validate numerous solutions. The TON-IOT data collection overcomes all of these shortcomings.

1.2 TON IOT Dataset

The TON-IOT data set, which includes information from the IOT, Linux, Windows, and the network, is suggested for use in the collection and evaluation of IOT data. This data set is described as having organized data in CSV format. It was developed in 2019 for Industry 4.0 (IOT) and Industrial IOT (IIOT). This data set includes a column that indicates if an attack occurred or not. The attacks are divided into nine categories: DDoS, DoS, ransomware, backdoors, injection, XSS, MITM, password cracking, and scanning.

A dataset for the Internet of Things (IoT) is divided into five categories: raw datasets, processed datasets, train-test datasets, description-stats datasets, and security events-ground truth datasets. In this project, we used the Train-Test datasets, which are divided into four folders: the Train-Test Network dataset, the Train-Test IoT dataset, and the Train-Test Linux dataset. Again, each folder is made up of CSV-formatted sub-files.

The Train-Test-Iot dataset contains five files:

1) **Train-Test-Iot-Modbus**: There are nine columns in this file: ts, date, time, There are 51106 records in the FC1-Read-Input-Register, FC2-Read-Discrete-Value, FC3-Read-Holding-register, and FC4-Read-Coil label types. We used the columns FC1-Read-Input-Register, FC2-Read-Discrete-Value, FC3-Read-Holding-Register, and FC4-Read-Coil for training data and type for testing data from the aforementioned columns. This data collection contains five different types of attacks: injection, backdoor, password, XSS, and scanning.

2) **Train-Test-Iot-Garagedoor**: This file has 59587 records and seven columns, including ts, date, time, door-state, sphone-signal, label, and type. For the training data, we took into account the door-state, phone-signal, and type columns from the aforementioned columns. This dataset contains seven different types of attacks: DDoS, backdoor, injection, password, ransomware, XSS, and scanning.

3) **Train-Test-GpsTracker**: There are 58960 records in this file, which have seven columns, including date, time, latitude, longitude, label, and type. We took into account latitude and longitude from the aforementioned fields for training data and type columns for testing data. This dataset contains seven different types of attacks, including DDoS, backdoor, injection, password, ransomware, XSS, and scanning.

4) **Train-Test-Thermostat**: This file contains seven columns namely ts, date, time, current-temperature, thermostat-status, label, and type, and consists of 52774 records. From the above columns, we considered the current-temperature, thermostat status for training data, and type column for testing data. There are six kinds of attacks present in this dataset namely backdoor, injection, password, ransomware, XSS, and scanning.

5) **Train-Test-Weather**: This file contains eight columns namely ts, date, time, temperature, pressure, humidity, label, and type, and consists of 59260 records. From the above columns, we considered temperature, pressure, and humidity for training data and type as testing data. There are seven kinds of attacks present in this dataset namely DDoS, backdoor, injection, password, ransomware, XSS, and scanning.

Similarly, Train-Test Linux contains mainly three files:

1) Linux Process Dataset:

In this particular file ts, PID, TRUN, TSLPI, TSLPU, POLI, NICE, PRI, RTPR, CPUNR, Status, EXC, State, CPU, CMD, label, and type, and it contains 160112 records present in the dataset. There is a total of eight attacks present namely, dos, DDoS, injection, mitim, normal, password, and scanning.

2) Linux Memory Dataset:

In this particular file ts, PID, MINFLT, VSTEXT, VSIZE, RSIZE, VGROW, RGROW, MEM, CMD, label type as columns, and the data set contains a total of 140112 records. There are 6 kinds of attacks used in the data set dos, DDoS, injection, mitm, normal, and password.

3) Linux Disk Data sets:

In this file column names such as ts, PID, RDDSK, WRDSK, WCANCL, DSK, CMD, label and the whole data set contains a total of 160112 records. There are mainly 8 attacks used in the data set namely DDoS, dos, injection, mitm, password, normal, scanning, and XSS.

Another dataset called **network dataset** is also used.

In this network dataset, ts, src-ip, src-port, dst-ip, dst-port, proto, service, duration, src-bytes, dst-bytes, conn-state, missed-bytes, src-pets, src- ip_bytes, dst-pkts, dst-ip-bytes, dns-query, dns-qclass, dns-qtype, dns- rcode, dns-AA, DNS-RD, dns-RA, DNS-rejected, SSL-version, SSL-cipher, ssl- resumed, ssl-established, ssl-subject, SSL-issuer, HTTP-trans-depth, HTTP- method, http-uri, HTTP-version, HTTP-request-body, http-response-body-len, HTTP-status-code, http-user-agent, http-orig-mime-types, HTTP-resp-mime-types, weird-name, weird-addl, weird-notice, label, type are the columns and consists of 461043 records.

We have chosen **Train-Test Linux** which includes Linux memory, Linux process datafiles and **Train-Test Network** which have data based on network datasets for this project.

2. Methodology

- Initially, we used the Panda library to load the dataset.
- We then removed the extra columns from the data frame.
- Using the label encoder, we then encoded the columns that don't include integers or floating-point numbers.
- Following encoding, we separated the columns into input ("x") and target ("y") fields.
- Next, we divided the data set into training and testing groups with a ratio of 70 to 30 each.
- The accuracy, recall, and F1 score for each machine learning model were later determined after we had sent the training and testing data to the various models.

We have calculated the evaluation metrics for both multiclass classification and binary class classification. We have plotted the graph for the precision-recall curve for the binary class classification.

3. Naïve Bayes

Due to attribute independence, Naive Bayes is one of the most effective data mining techniques. Due to the fact that Naive Bayes treats each character as an independent variable, this model is thought to be efficient. This can be applied in real-world settings and is particularly effective in supervised learning. The main advantage of this method is that it just requires a small amount of the training data required for classification.

The posterior probability, commonly known as the algorithm to determine the likelihood that an event will depend on the likelihood that its prior event will occur, is calculated as follows:

$$P(A|B) = P(B|A) * P(A) / P(B)$$

Gaussian Naïve Bayes:

In [4]:
```
gnb=GaussianNB()
model=gnb.fit(x_train,y_train.values.ravel())
y_pred=model.predict(x_test)

print("Accuracy: ",accuracy_score(y_test,y_pred))
print("Recall: ",recall_score(y_test,y_pred,average='weighted'))
print("F1score: ",f1_score(y_test,y_pred,average='weighted'))
```

4. Random Forest

The Random Forest Classifier has seen widespread use in classification, prediction, and regression. They are combined with a series of tree-structure classifiers.

In the Random Forest Classifier margin function is calculated to find out the accuracy. The larger the margin value, the higher the accuracy of the classification prediction.

It is an ensemble method that is based on the divide-and-conquer approach of decision trees.

Implementation:

```
In [5]: rfc=RandomForestClassifier(n_estimators=100, random_state=42, max_depth=10)
        rfc.fit(x_train,y_train.values.ravel())
        y_pred=rfc.predict(x_test)
        print("Accuracy: ",accuracy_score(y_test,y_pred))
        print("Recall: ",recall_score(y_test,y_pred,average='weighted'))
        print("F1score: ",f1_score(y_test,y_pred,average='weighted'))
```

5. XG Boost

XG Boost classifier which stands for Extreme Gradient Boosting, in which decision trees are created in sequential form. Weights are given to all the independent variables and are then given to the second decision tree. These classifiers are then ensemble to give an accurate model.

It can work on regression and classification problems. It combines several models into a single model to correct the errors made by existing models.

Implementation:

In [6]:
```
xgb_classifier=xgb.XGBClassifier(n_estimators=100, random_state=42, max_depth=10)
xgb_classifier.fit(x_train,y_train.values.ravel())
y_pred=xgb_classifier.predict(x_test)

print("Accuracy: ",accuracy_score(y_test,y_pred))
print("Recall: ",recall_score(y_test,y_pred,average='weighted'))
print("F1score: ",f1_score(y_test,y_pred,average='weighted'))
```

6. MLP Classifier

Multi-Layered Perceptron [MLP] Classifier is proposed for fault detection which is inexpensive, reliable, and non-invasive, which maps input data sets to a set of appropriate outputs. It consists of multiple layers and every layer is connected to its forwarding neighbor. MLP Classifier trains continuously at loop since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

Implementation:

```
In [7]: model = MLPClassifier(max_iter=100, random_state=42)
        model.fit(x_train, y_train.values.ravel())
        y_pred=model.predict(x_test)

        print("Accuracy: ",accuracy_score(y_test,y_pred))
        print("Recall: ",recall_score(y_test,y_pred,average='weighted'))
        print("F1score: ",f1_score(y_test,y_pred,average='weighted'))
```

7. Decision Tree

A Decision Tree Classifier is one of the simplest algorithms that have the ability to convert complex decision-making problems into a simple process. As per the results, it is said that the decision tree classifier has great potential in solving pattern recognition problems.

The decision Tree Classifier is depicted in a flowchart-like tree structure in which each internal corresponds to a test on an attribute, each branch represents a leaf node. When a tuple Y is considered, the values of that Y tuple are tested against the tree and a new path as a result is shown which holds the prediction data for that tuple.

Implementation:

In [10]:
```
clf=DecisionTreeClassifier()
clf.fit(x_train,y_train.values.ravel())
y_pred=clf.predict(x_test)
print("Accuracy: ",accuracy_score(y_test,y_pred_mlp))
print("Recall: ",recall_score(y_test,y_pred_mlp,average='weighted'))
print("F1score: ",f1_score(y_test,y_pred_mlp,average='weighted'))
```

8. K Nearest Neighbor

K Nearest Neighbor is a learning model which is known for its simplicity. In the KNN algorithm, whenever data has to get clarified, the training data is used to classify the data. Whenever new data has to get labeled KNN operates over similar data points and results in the nearest point available.

The classification of the data available is figured out by measuring out the distance between the test sample and the training sample to get the final output. K value represents the number of nearest neighbors present in the data set.

The Euclidean formula for finding the distance between two points is as follows:

$$\sqrt{\{(x-a)\ ^\ \{2\} + (y-b)\ ^\ \{2\}\}}$$

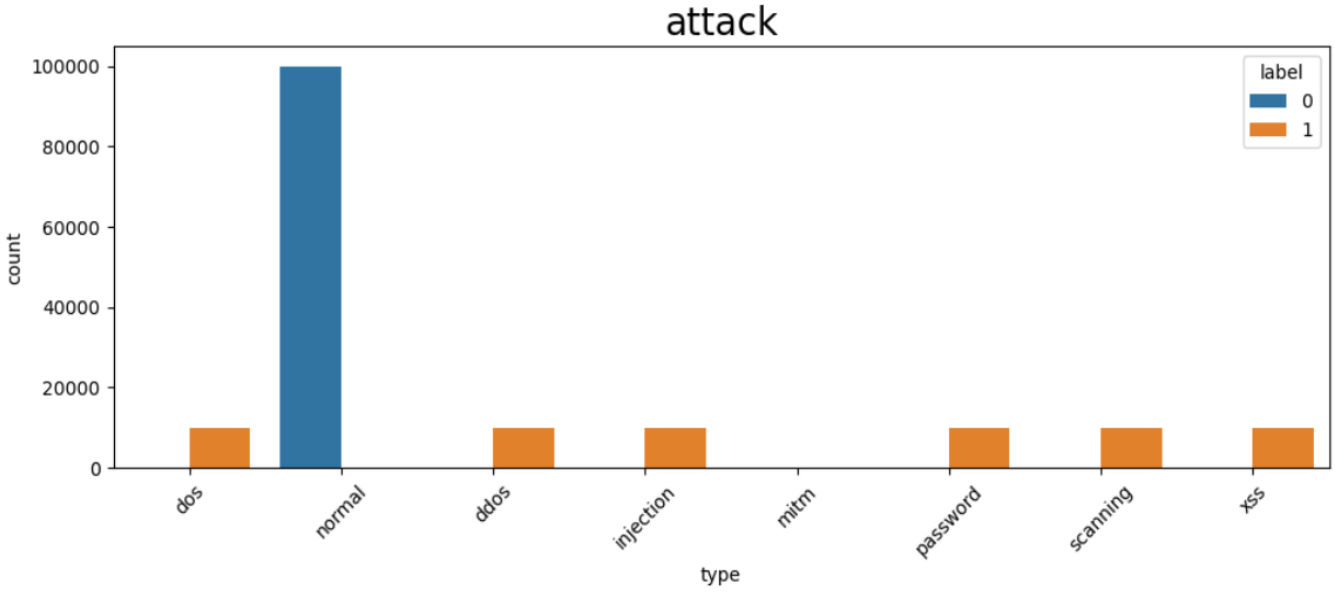
Implementation:

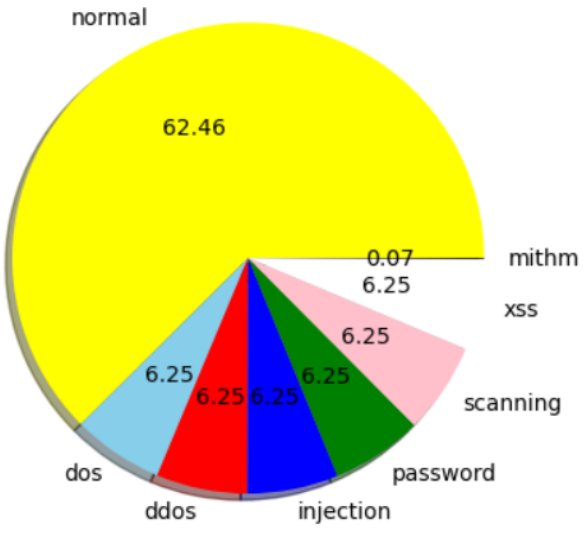
```
In [11]: classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
         classifier.fit(x_train,y_train.values.ravel())
         y_pred=classifier.predict(x_test)
         print("Accuracy: ",accuracy_score(y_test,y_pred_mlp))
         print("Recall: ",recall_score(y_test,y_pred_mlp,average='weighted'))
         print("F1score: ",f1_score(y_test,y_pred_mlp,average='weighted'))
```

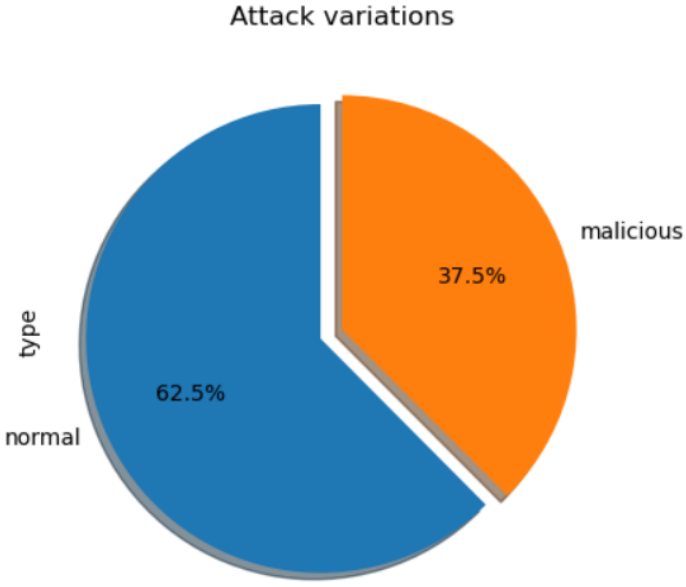
9. Results

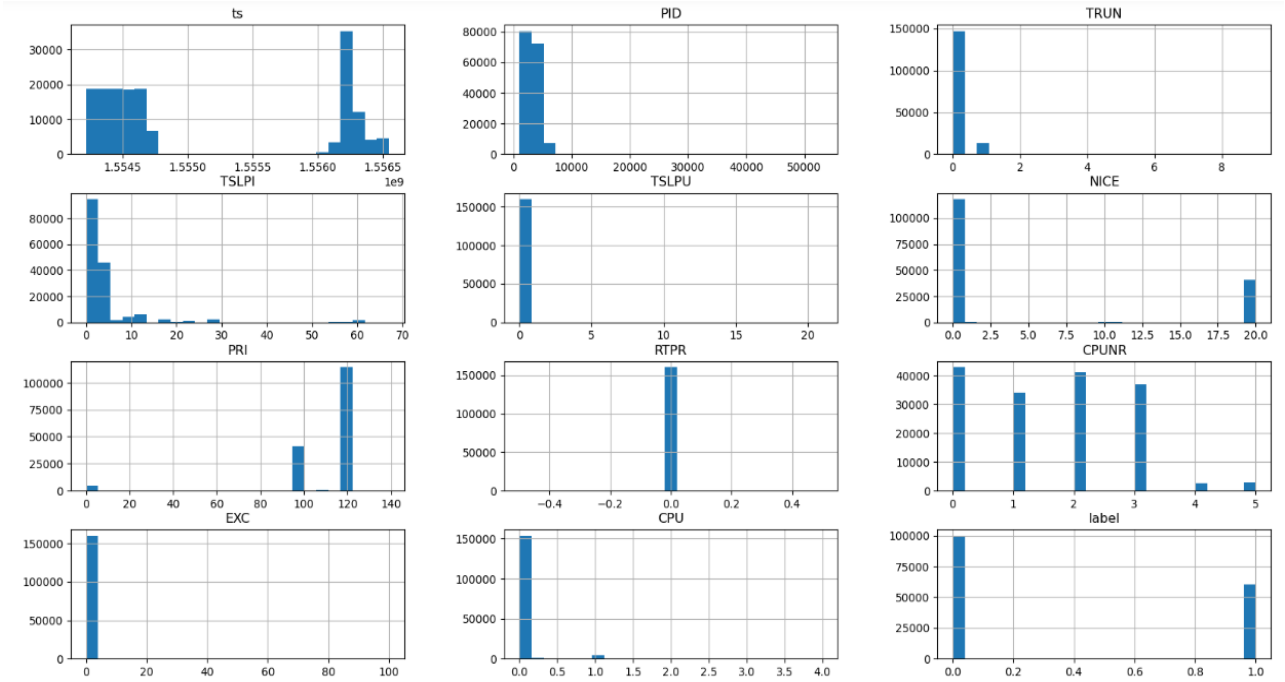
Linux Process

1. Exploratory Data Analysis

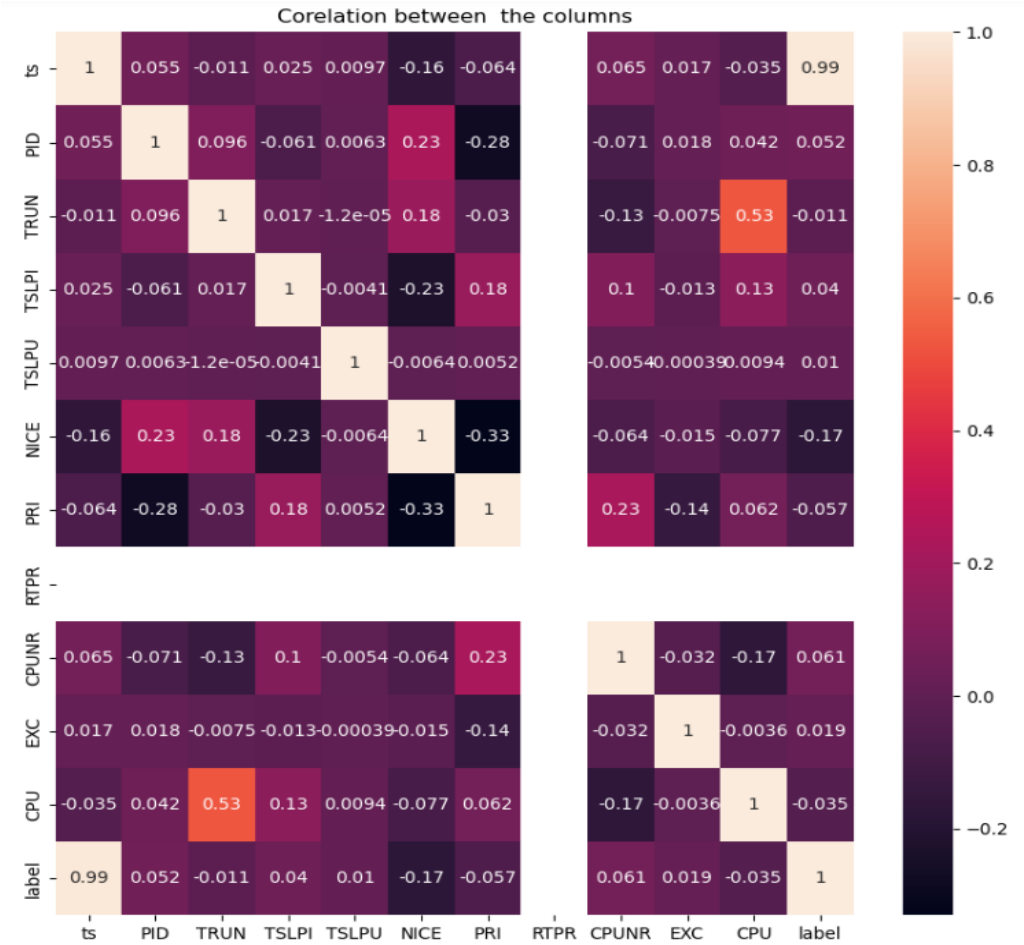






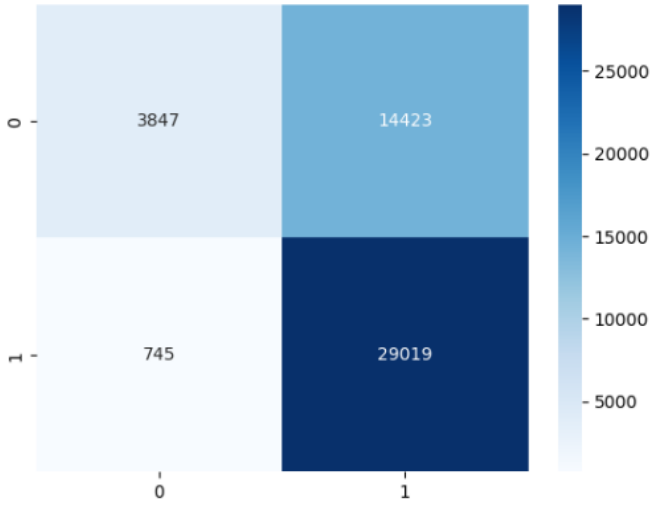


2. Correlation of the columns

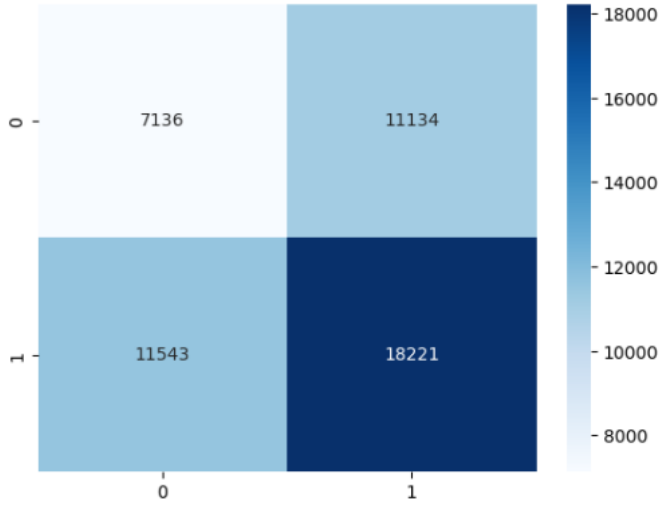


3. Confusion matrices

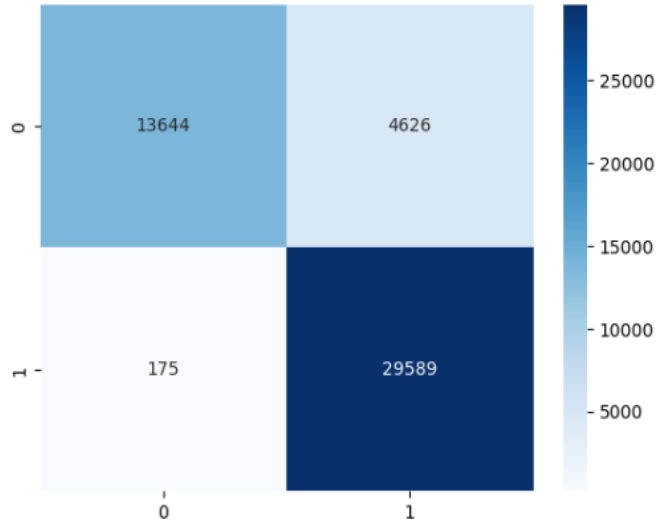
Gaussian NB

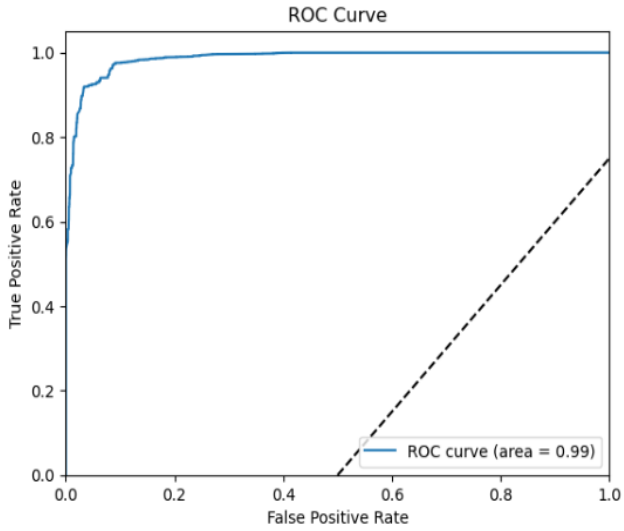


Multinomial NB

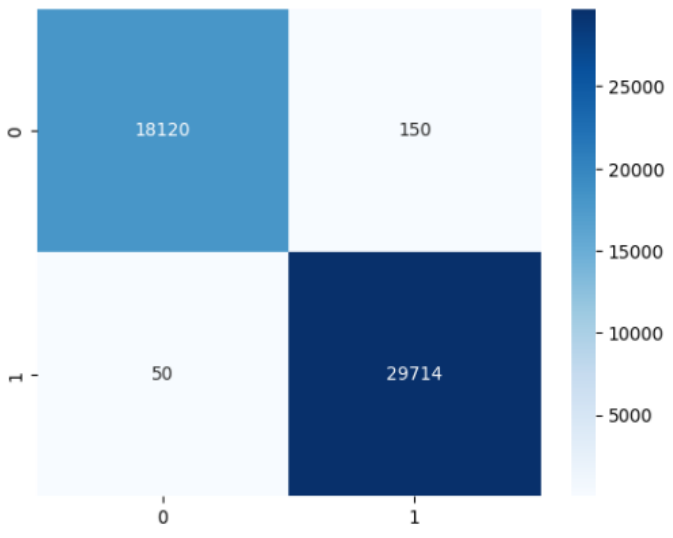


Random Forest Classifier

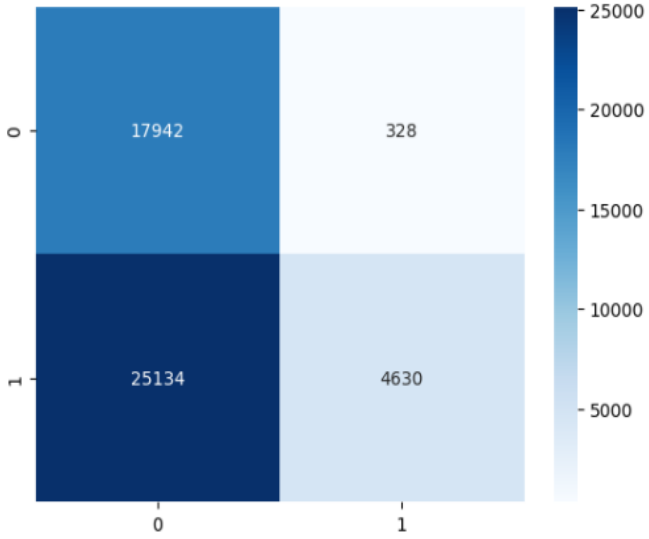




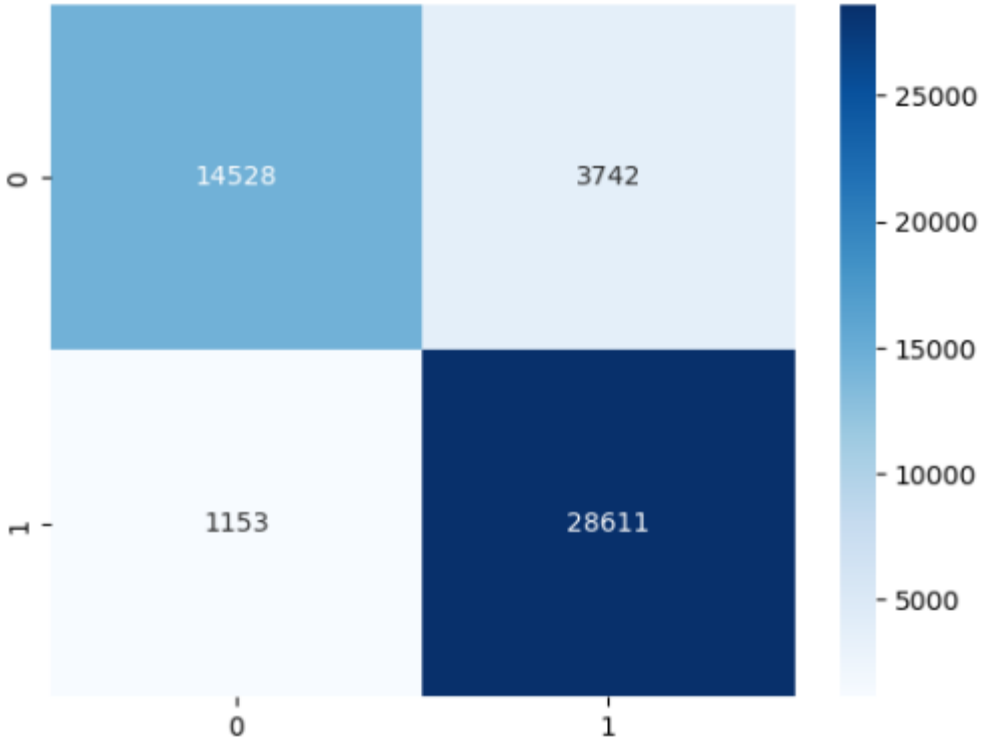
XGB Classifier



Multi-Level Perceptron



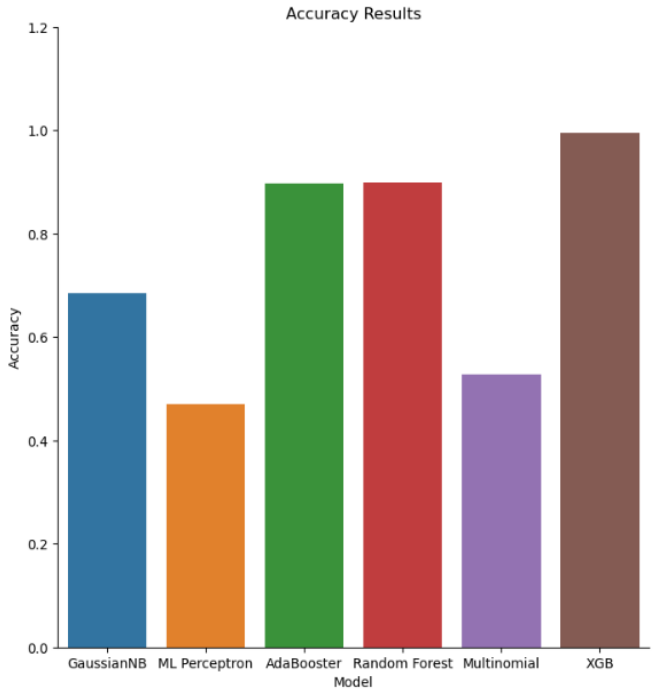
AdaBoost Classifier

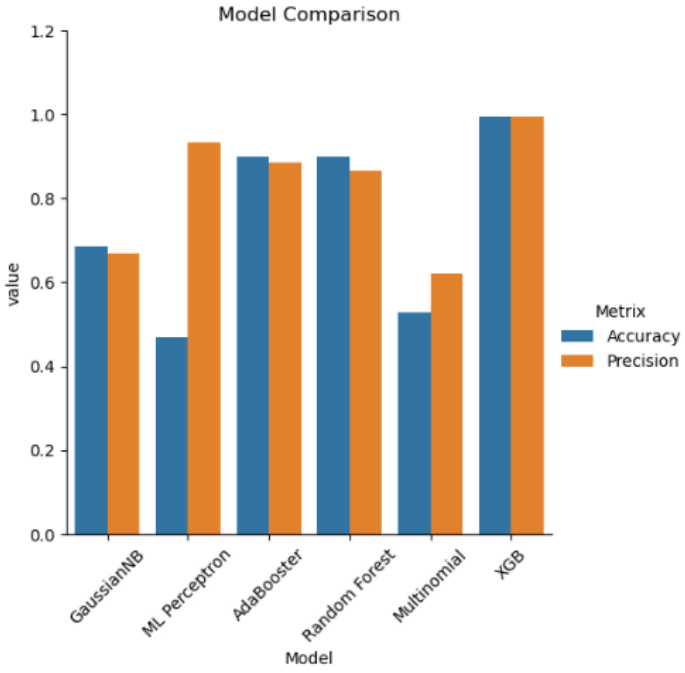


Accuracy & Precision Results

	Model	Accuracy	Precision
0	GaussianNB	0.684224	0.667994
1	ML Perceptron	0.469917	0.933844
2	AdaBooster	0.898093	0.884338
3	Random Forest	0.900050	0.864796
4	Multinomial	0.527897	0.620712
5	XGB	0.995836	0.994977

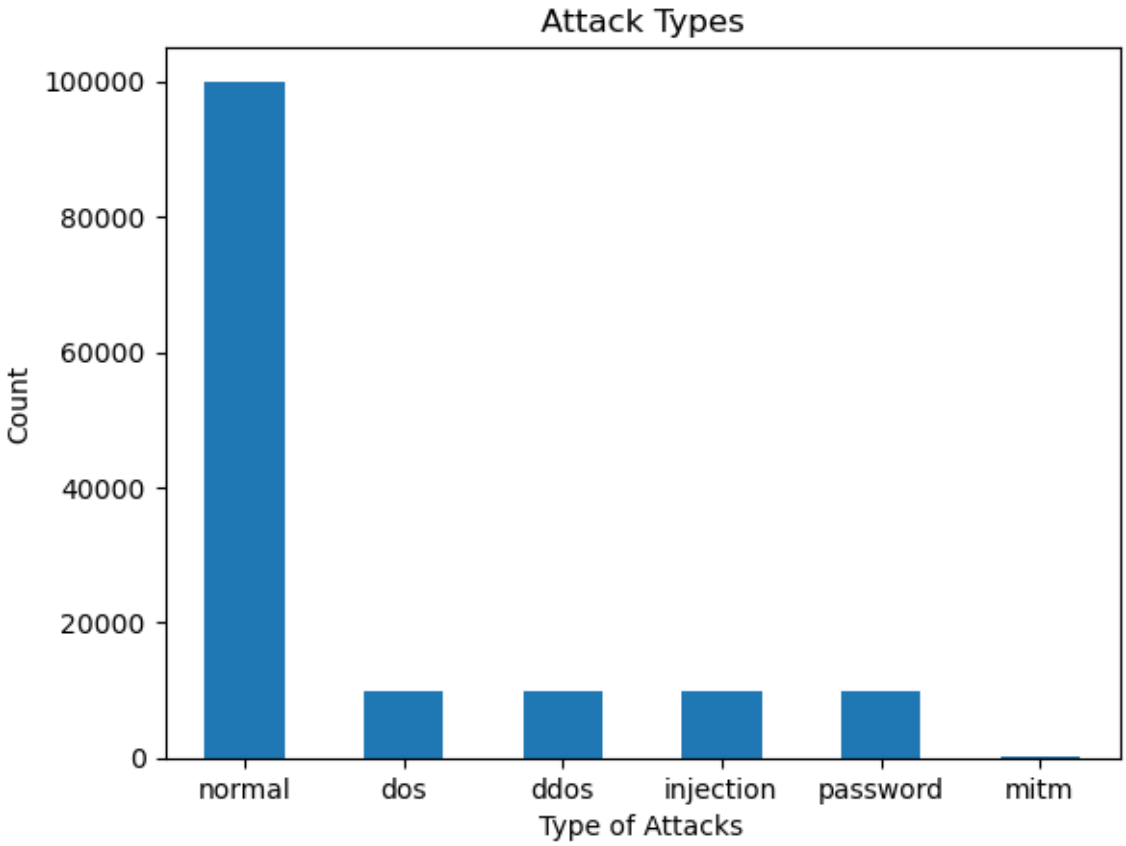
Models Comparison

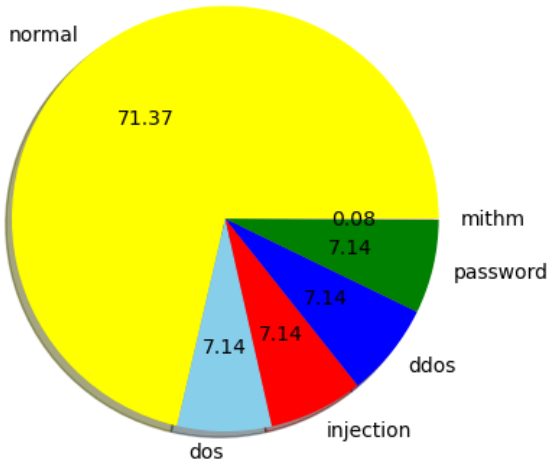


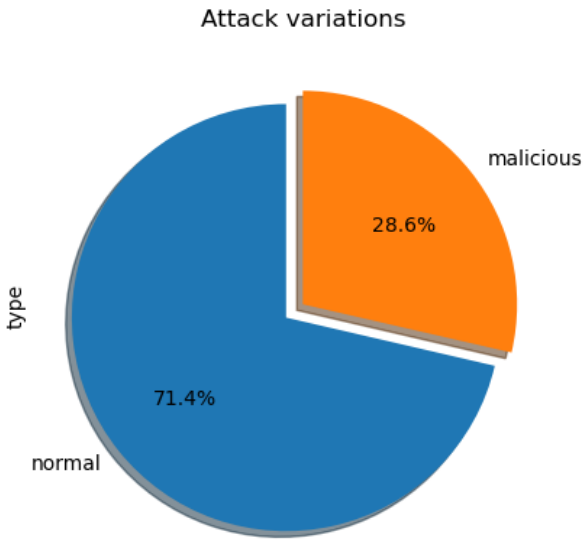


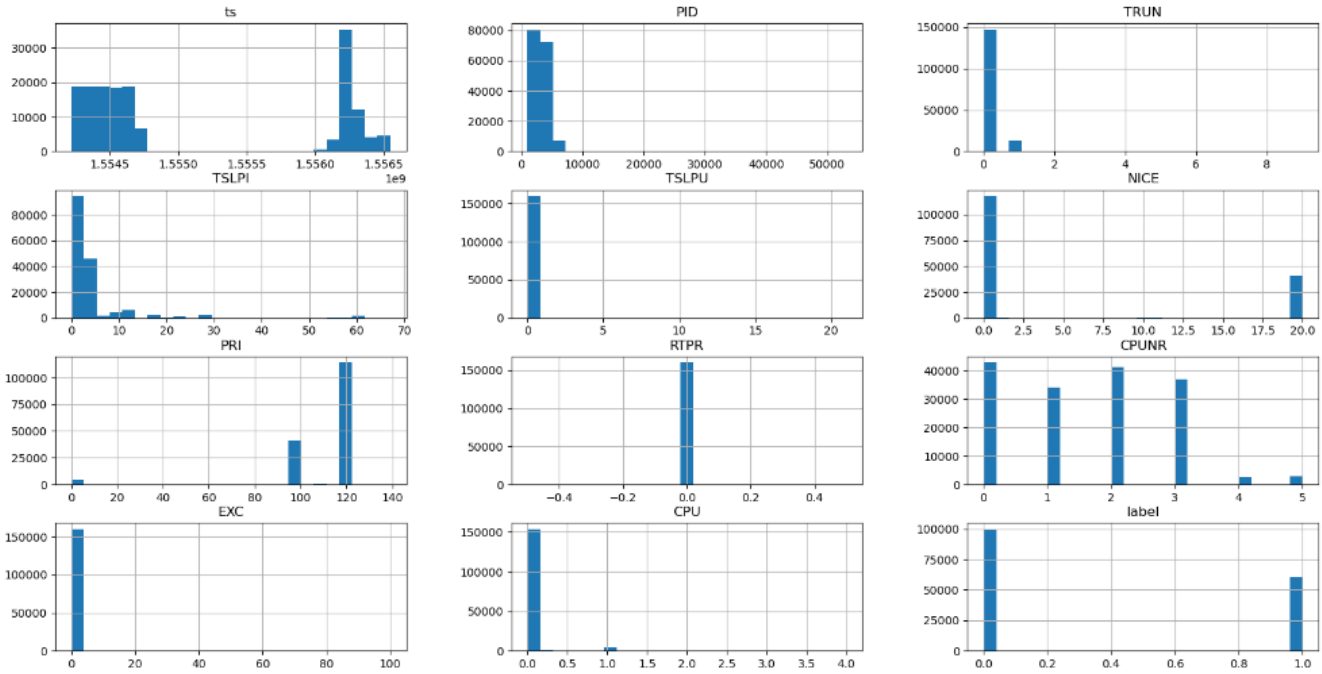
Linux Memory

1. Exploratory Data Analysis





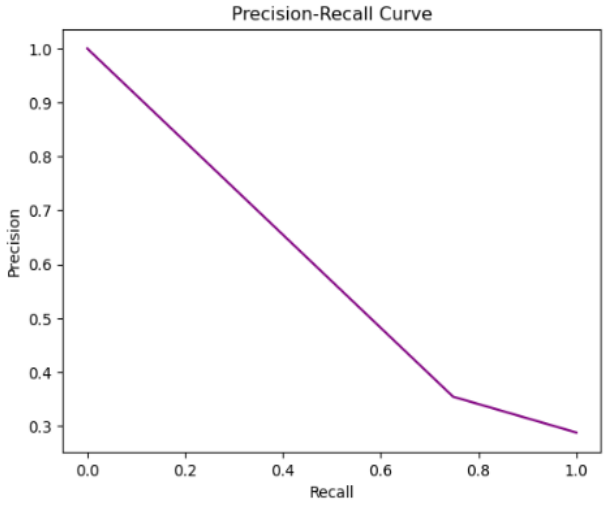




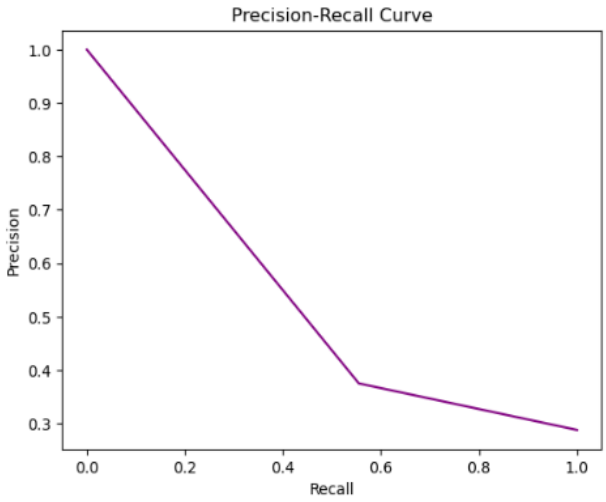
2. Correlation of the columns

3. Precision-Recall Curves & Confusion Matrix

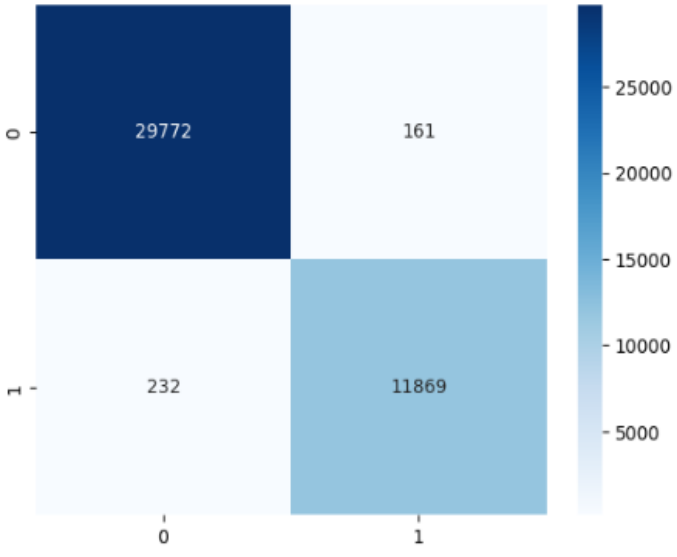
Gaussian NB

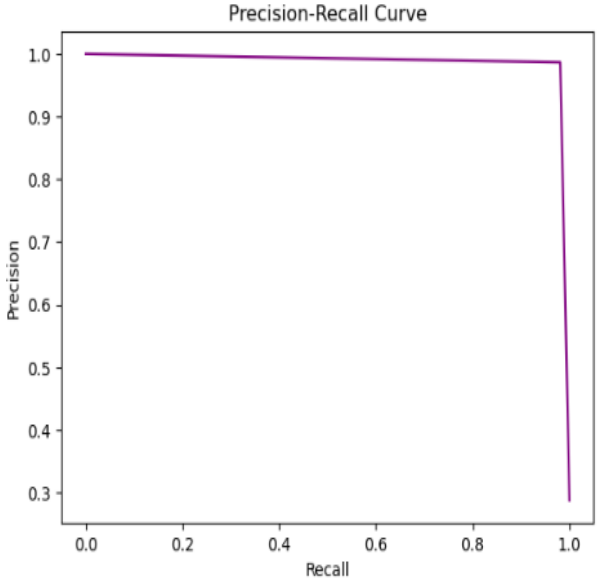


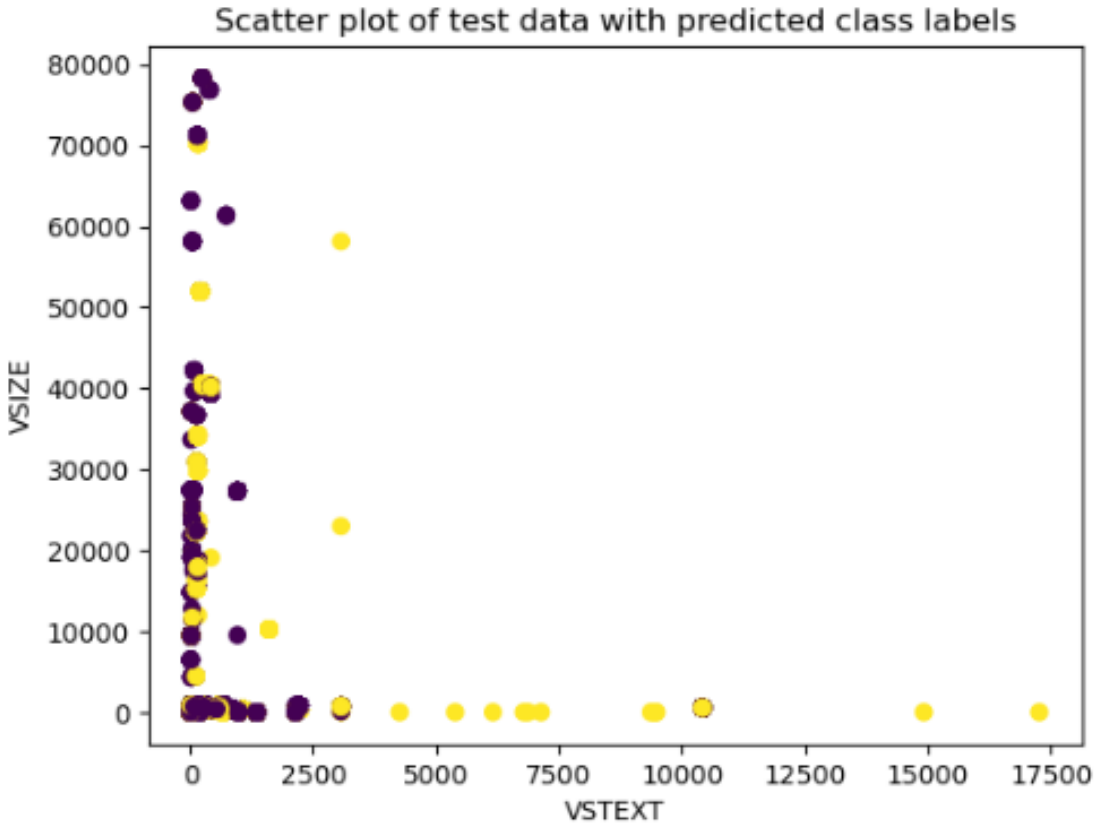
Multinomial NB



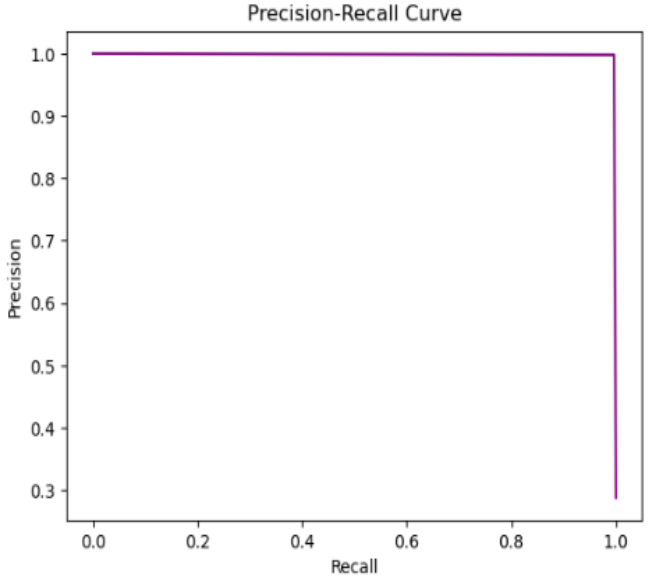
Random Forest Classifier

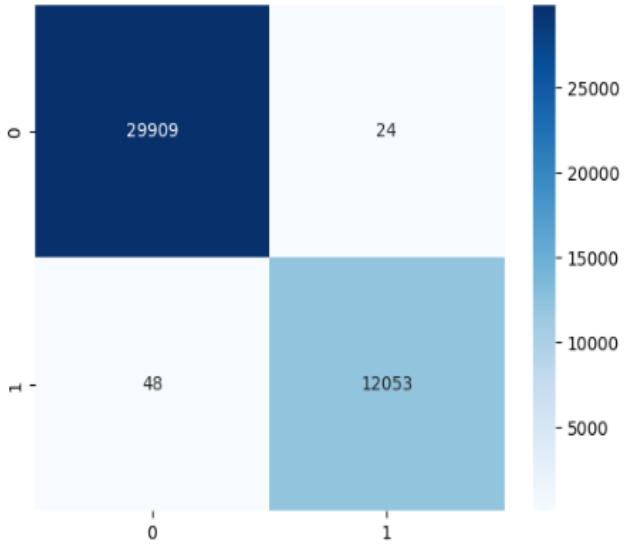




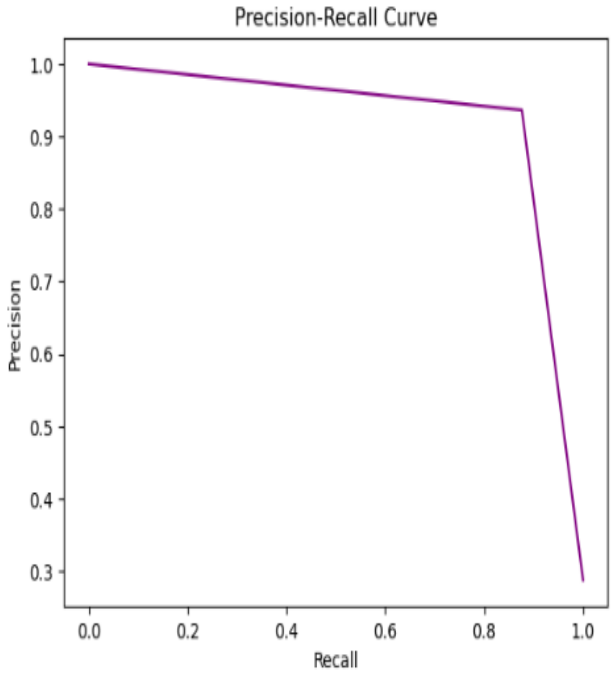


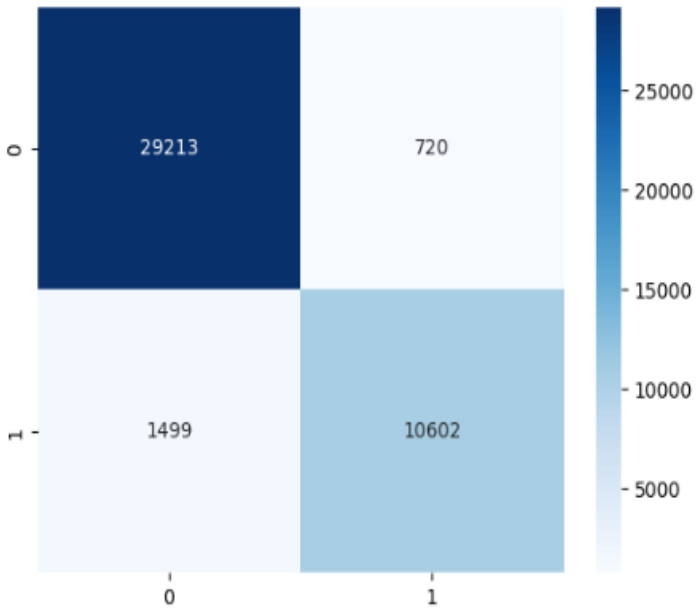
XGB Classifier



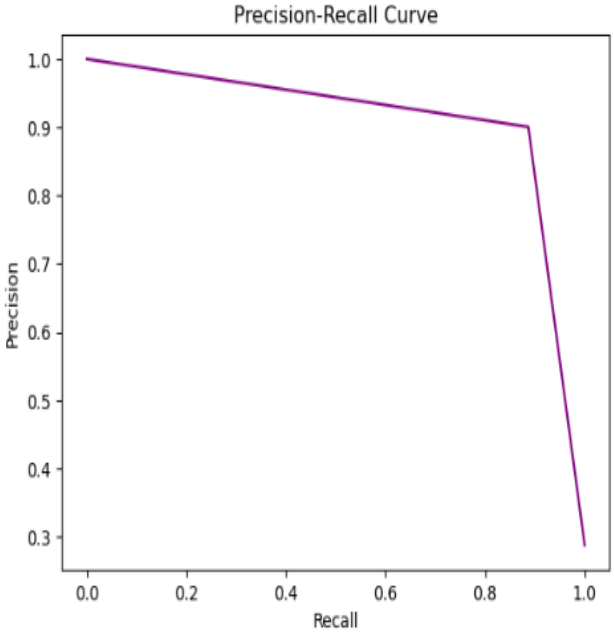


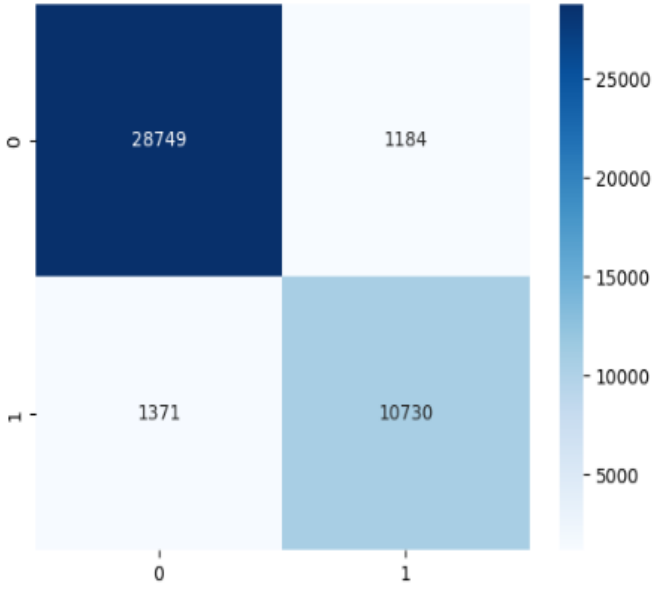
AdaBoost Classifier



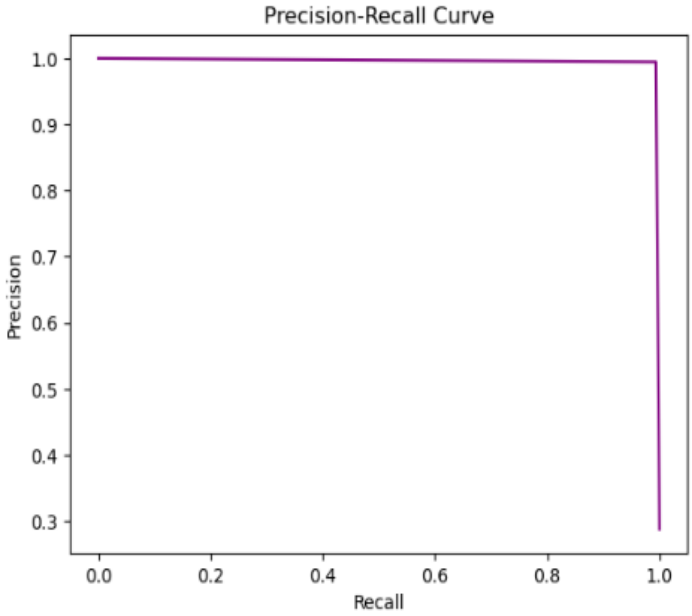


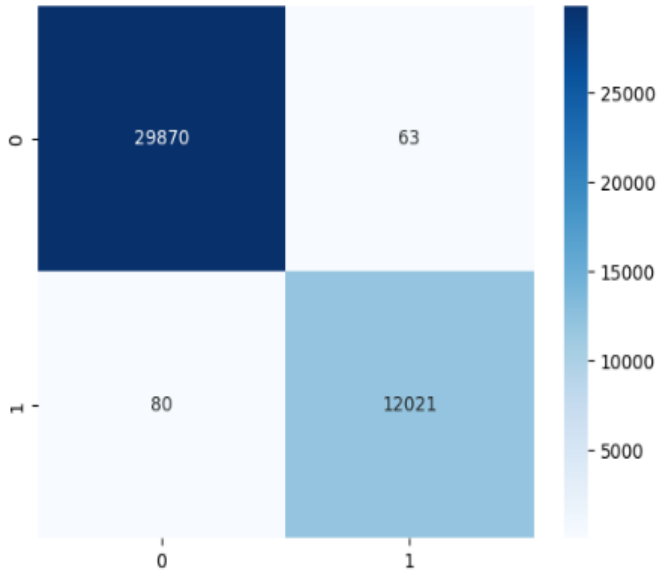
Multi-Level Perceptron



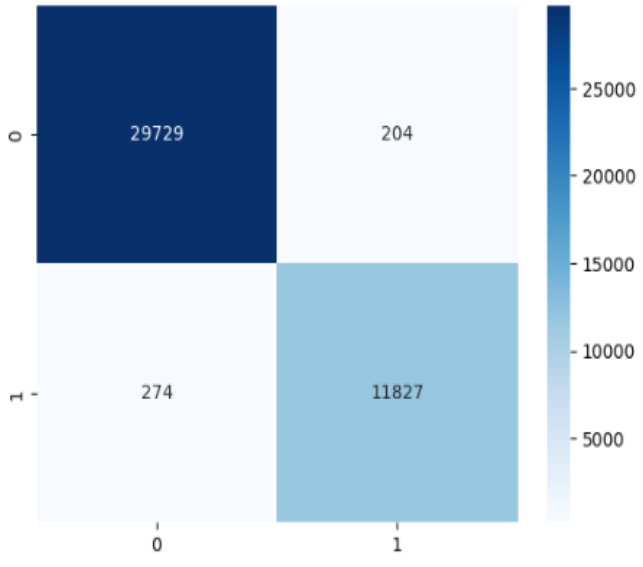


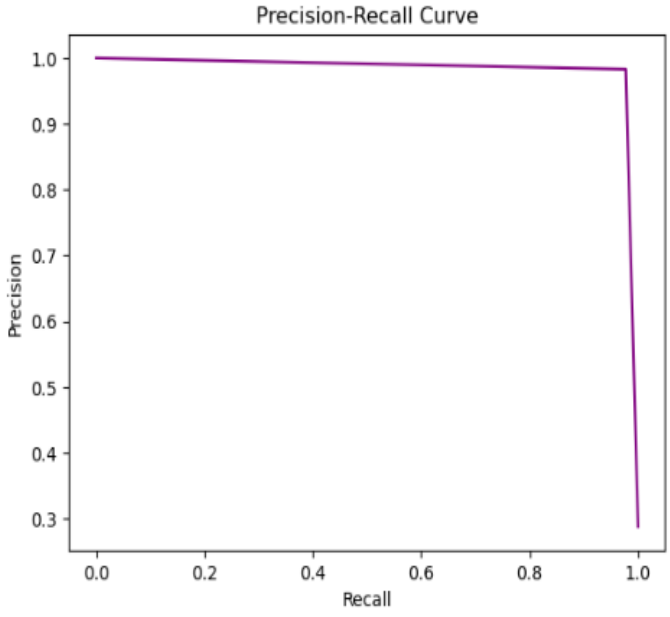
Decision Tree Classifier



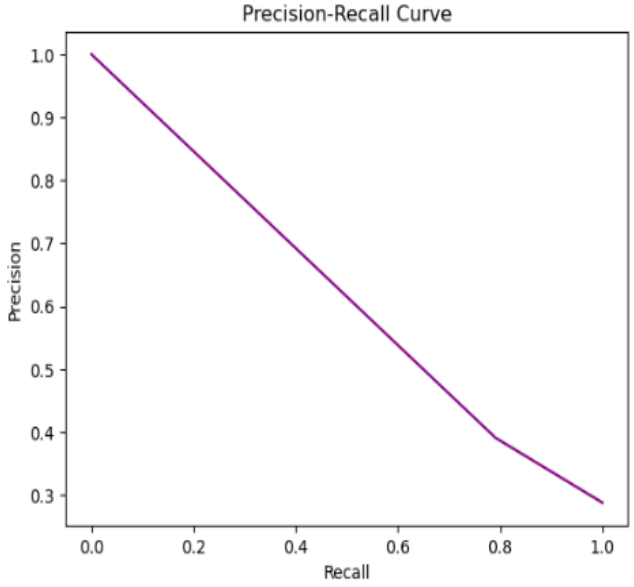


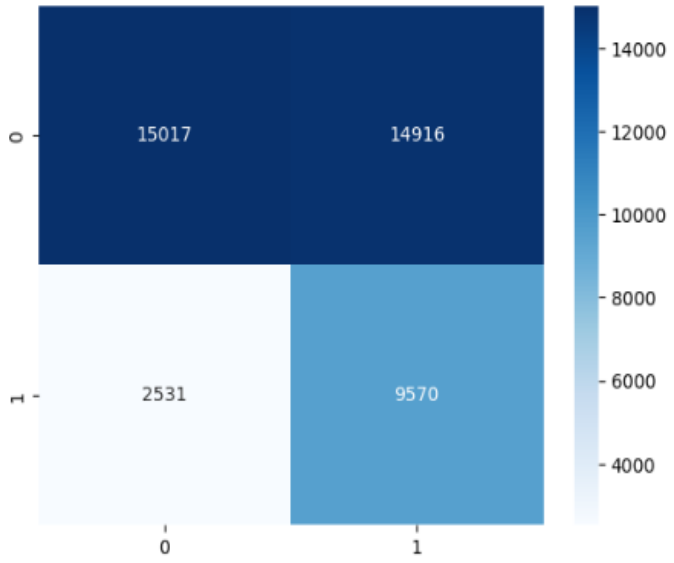
K Nearest Neighbor Classifier



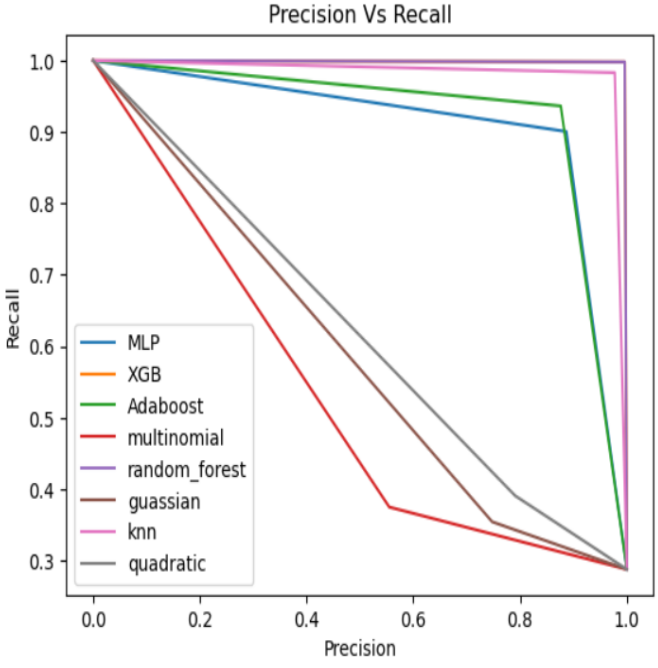


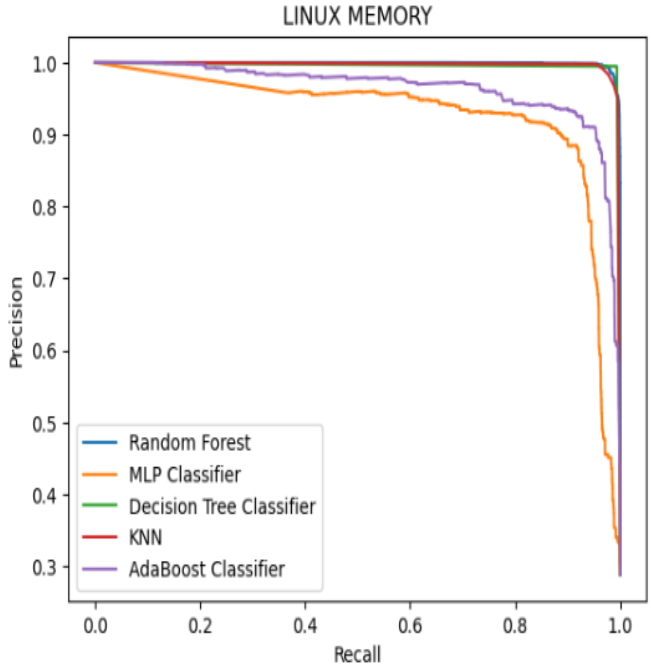
Quadratic DiscriminantAnalysis





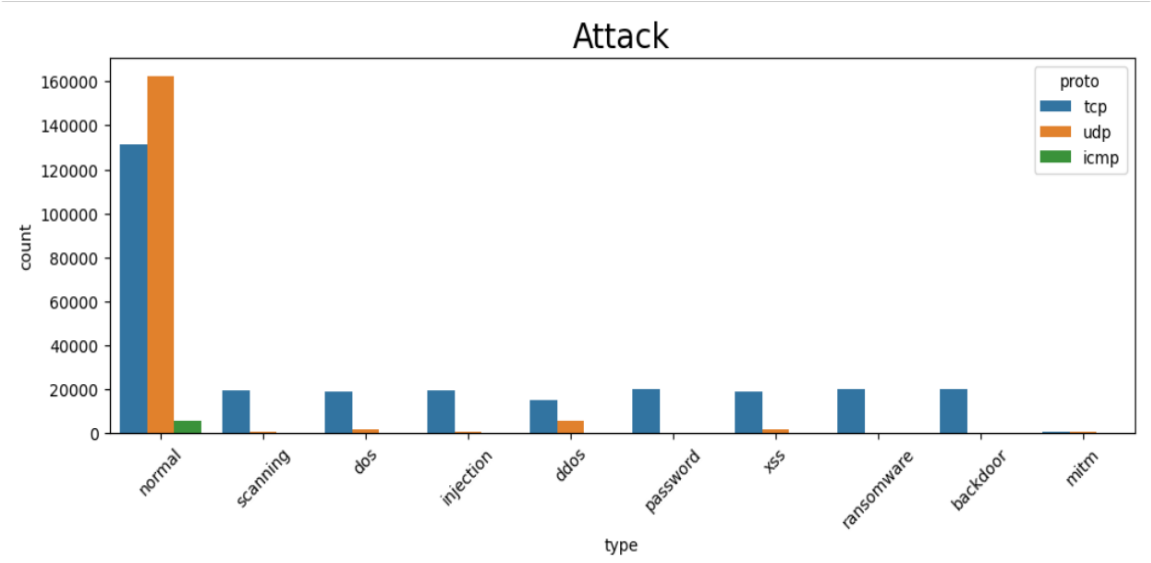
4. Precision-Recall of all Models

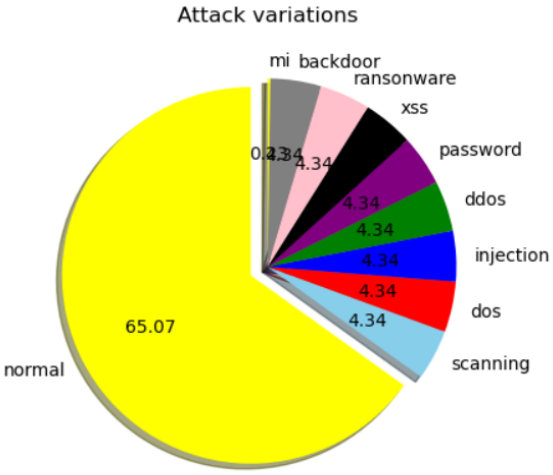


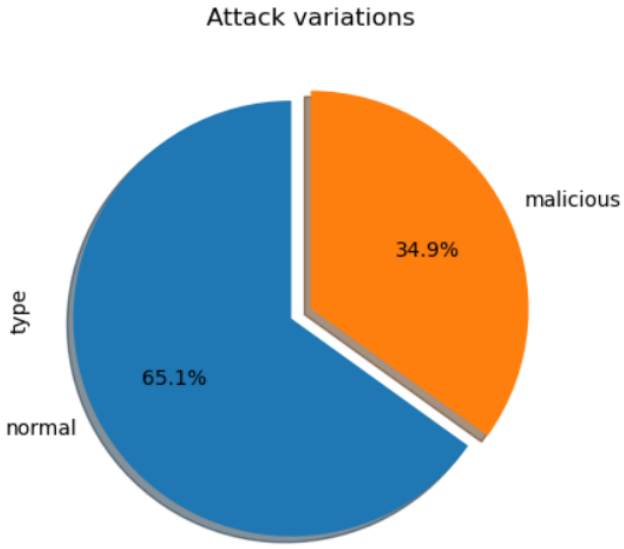


Network

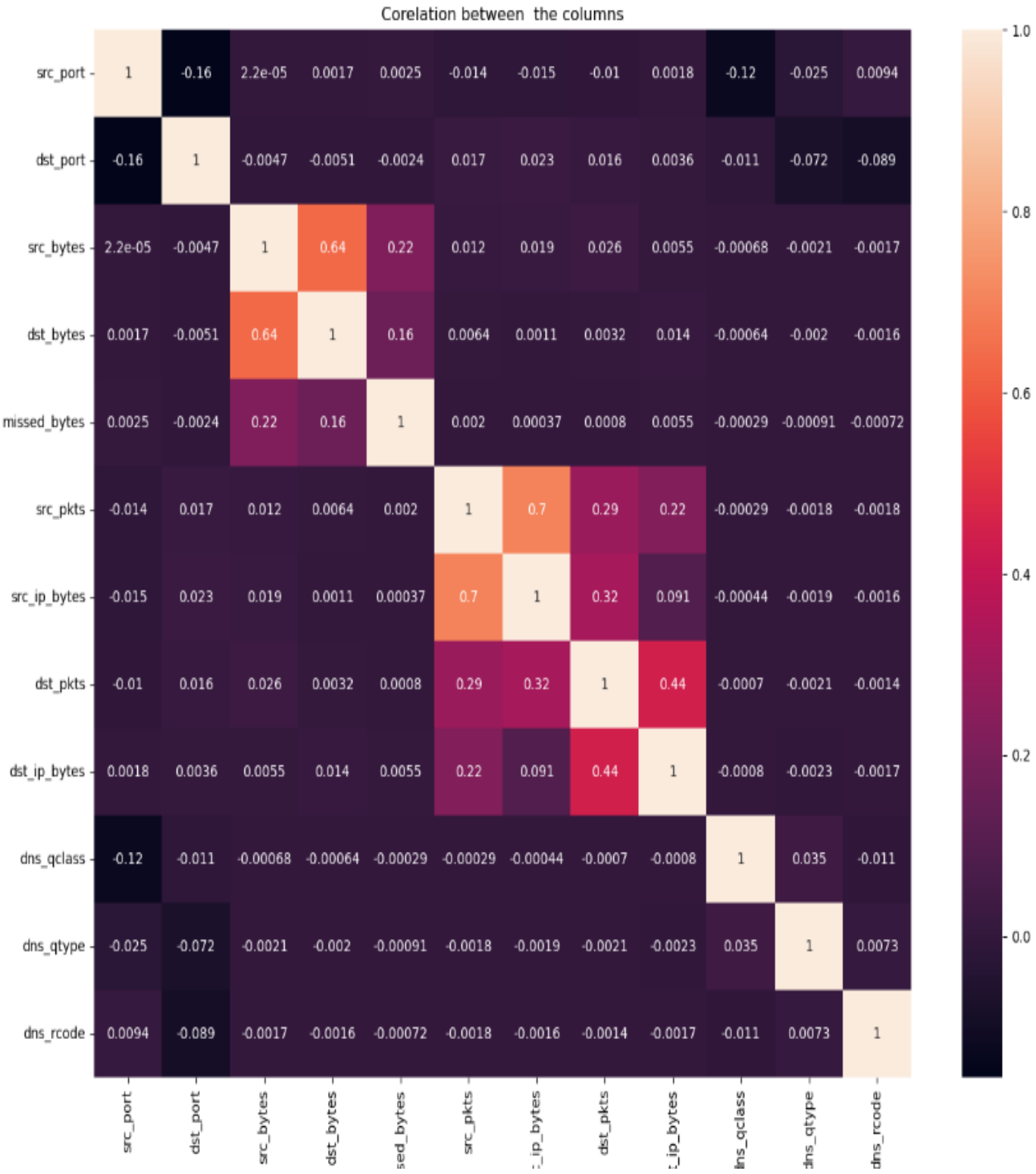
1. Exploratory Data Analysis





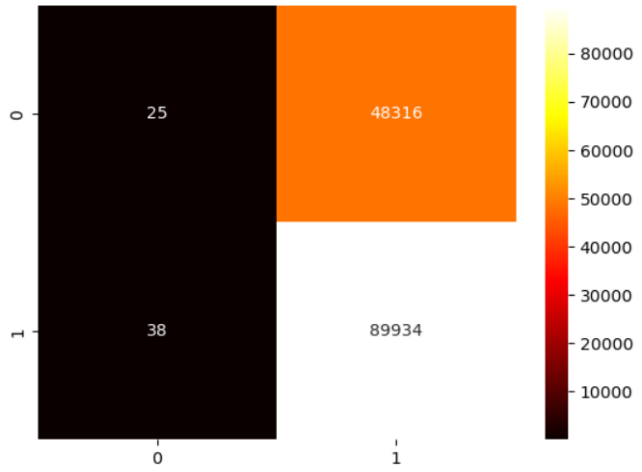


2. Correlation of the columns

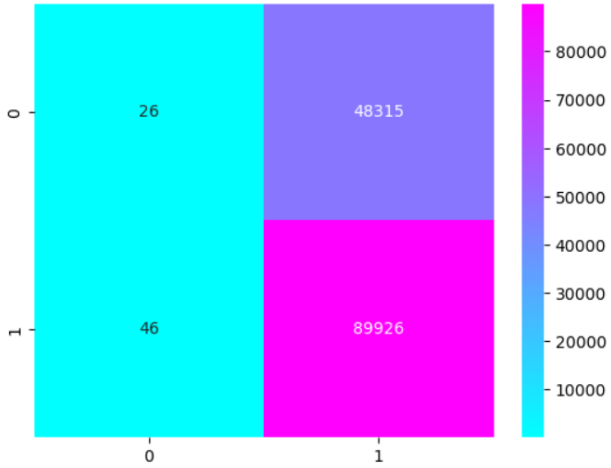


3. Confusion matrices

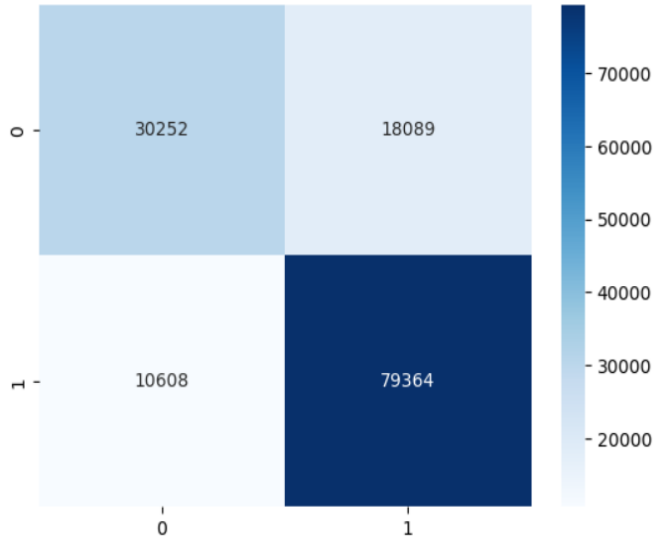
Gaussian NB



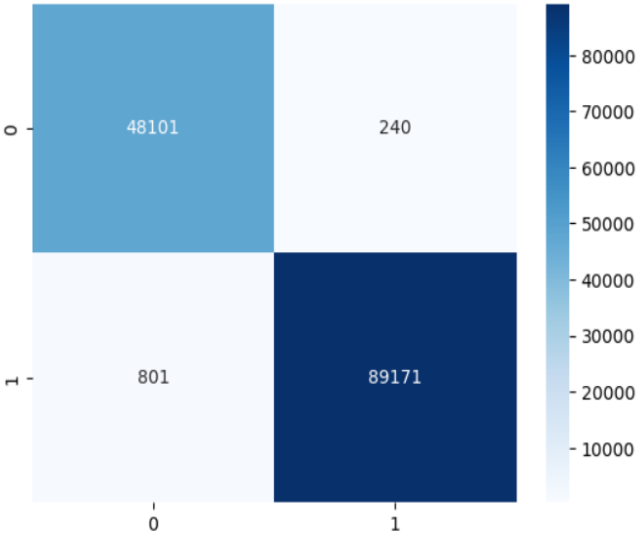
Multinomial NB



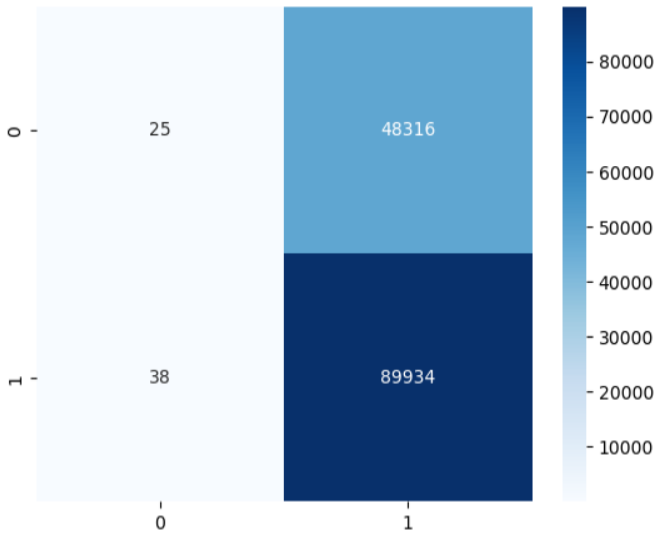
Bernoulli NB



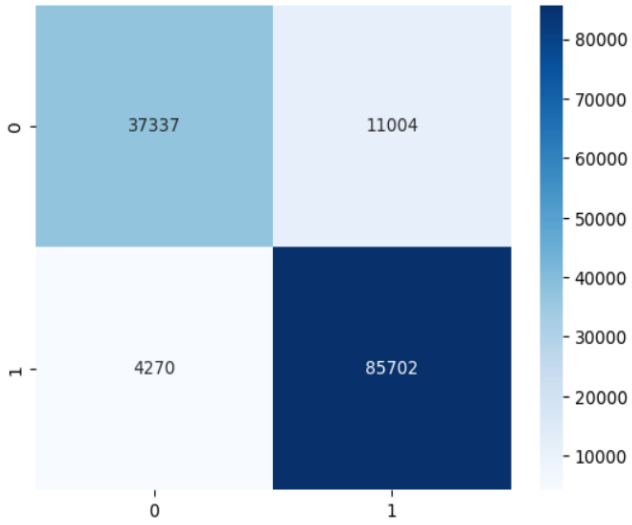
Random Forest Classifier



XGBoost Classifier

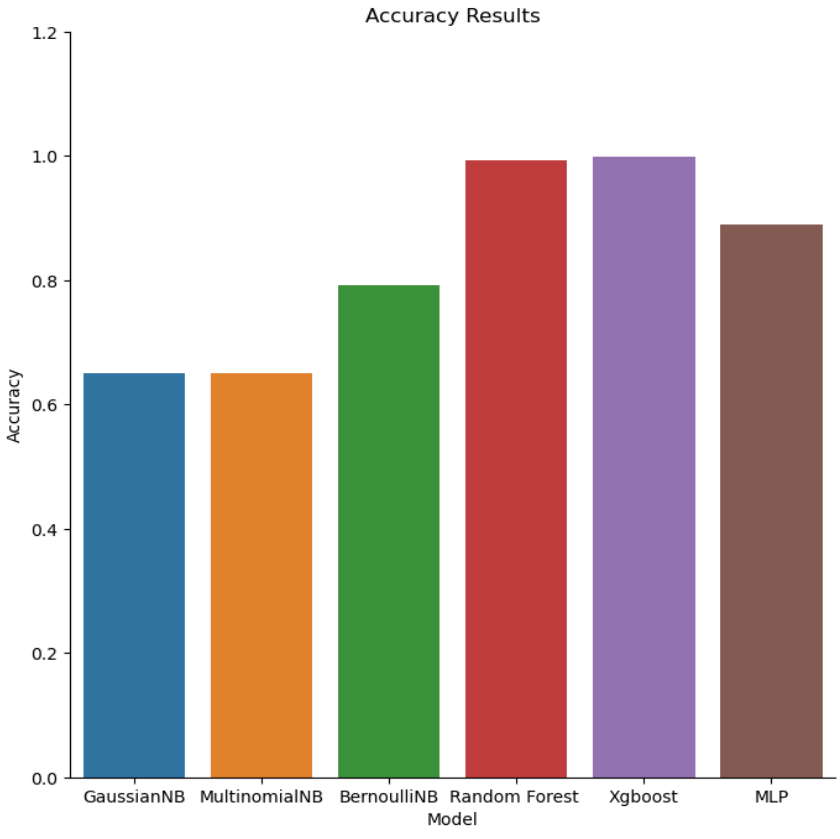


Multi-Level Perceptron



4. Model Comparison

	Model	Accuracy
0	GaussianNB	0.650402
1	MultinomialNB	0.650351
2	BernoulliNB	0.792521
3	Random Forest	0.992474
4	Xgboost	0.998373
5	MLP	0.889569



10. Conclusion

According to the stated research, discovering the breach is critical since the user's data could be leaked or an entire database system could be compromised if adequate precautions are not followed. Because the Internet of things is so important in modern technology, algorithms such as KNN, Xg Boost Classifier, Decision Tree Classifier, and others are used to assess the correctness of each Ton-IoT dataset. Based on the findings, we can conclude that the Xg Boost and KNN algorithms provided the best overall accuracy for binary and multi-class classification.

Similarly, several other algorithms can also be under various datasets, to find the accuracy, recall and F1 score to find the intrusions happening in the systems.

11. Future Work

In future we will perform hyper-tuning to increase the performance of the models and increasing the performance metrics of the models.

12. References

1. N. Moustafa, M. Ahmed, S. Ahmed, Data analytics-enabled intrusion detection: Evaluations of ton iot linux datasets, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, 2020, pp. 727–735.

2. Priyanka, D. Kumar, Decision tree classifier: A detailed survey, International Journal of Information and Decision Sciences 12 (3) (2020) 246–269.

3. A. Thakkar, R. Lohiya, A review of the advancement in intrusion detection datasets, Procedia Computer Science 167 (2020) 636–645.

4. M. Sarhan, S. Layeghy, M. Portmann, Towards a standard feature set for network intrusion detection system datasets, Mobile Networks and Applications 27 (1) (2022) 357–370.

5. N. Shone, T. N. Ngoc, V. D. Phai, Q. Shi, A deep learning approach to network intrusion detection, IEEE Transactions on emerging topics in computational intelligence 2 (1) (2018) 41–50

6. S. Zavrak, M. İskefiyeli, Anomaly-based intrusion detection from network flow features using variational autoencoder, IEEE Access 8 (2020) 108346–108358.

7. R. G. Bace, Intrusion detection, Sams Publishing, 2000.

8. S. Latif, Z. e Huma, S. S. Jamal, F. Ahmed, J. Ahmad, A. Zahid, K. Dashtipour, M. U. Aftab, M. Ahmad, Q. H. Abbasi, Intrusion detection framework for the internet of things using a dense random neural network, IEEE Transactions on Industrial Informatics (2021).