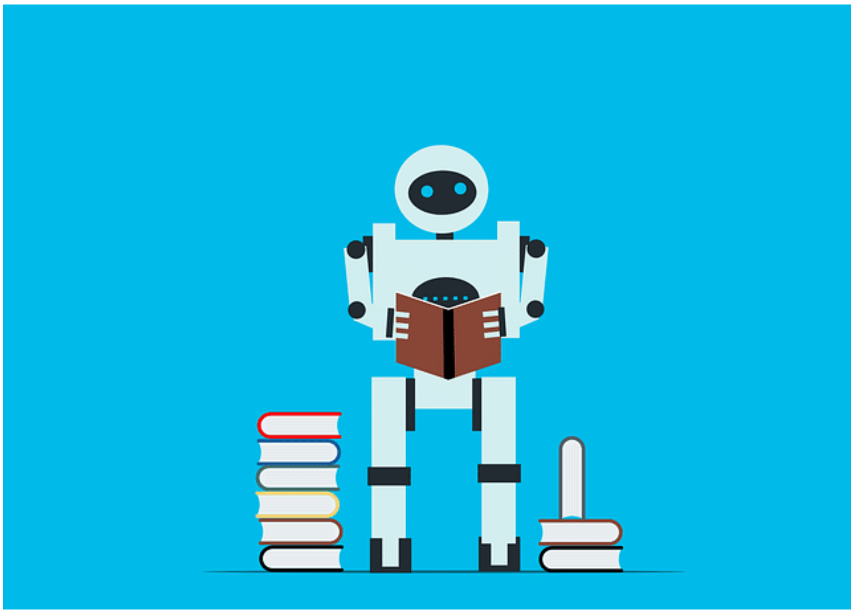
Report Machine Learning

Intrusion Detection with the UNSW-NB15 Dataset

Hugo Hajdarevic, Clément Garrigou, Quentin Gaidier – OCC2



December 12, 2025

Github-link

Contents

1 Business Case

Problem. In a more and more digitalized and dangerous world, cyberattacks have become a common issue for companies. To protect against these threats, tools are created to detect and prevent attacks from occurring.

Our dataset focuses on the impact of an intrusion on network activity. Analyzing the normal and under-attack behavior of a network can be a strong indicator of whether an intrusion has occurred or not.

The UNSW-NB15 dataset combines a hybrid of real modern normal activities and synthetic contemporary attack behaviors. This allows us to train machine learning models to detect an intrusion by analyzing key characteristics of network activity.

This topic is directly related to our major, **Connected Objects and Cybersecurity**, making it a very relevant project.

2 Dataset Description

The dataset used is the **UNSW-NB15** data set, a well-known benchmark for network intrusion detection. It contains real and synthetic network traffic with both normal connections and many types of cyberattacks.

Our goal is to build a machine learning model that can automatically identify whether a network connection is **normal** or **malicious** based on its features.

The dataset contains 45 columns describing each flow (duration, number of packets, bytes, protocol, flags, etc.) and one label indicating if the connection is an attack or not, as well as another label indicating the attack type, in the case of an attack.

3 Data Exploration

3.1 Initial Checks

We first checked the structure of the dataset (number of rows and columns) and verified that there are no missing values. As expected from the dataset description, there are no missing values, which means that the dataset is fully operational and ready for preparation.

3.2 Correlation Between `sbytes` and Label

To better understand the relationships, we start by focusing on the number of bytes sent (`sbytes`) and how it relates to the label.

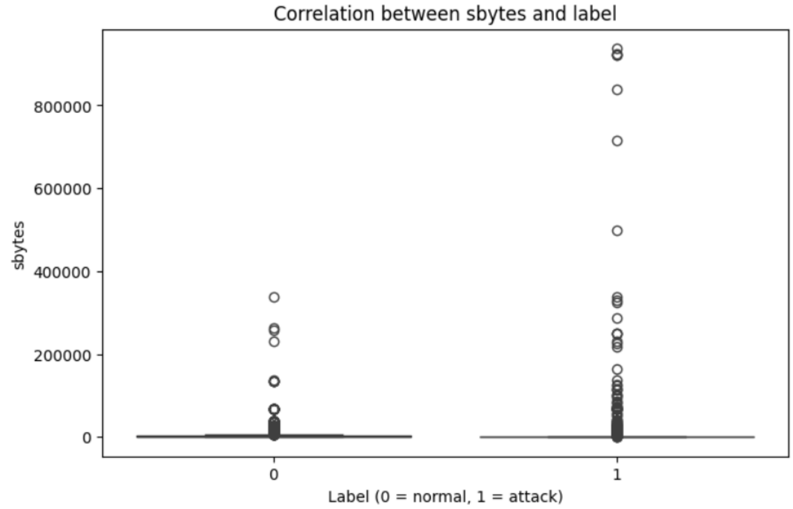


Figure 1: Correlation between sbytes and label

We cannot see a lot on this plot, so we will zoom in to get a more interesting view on our data, using another box.

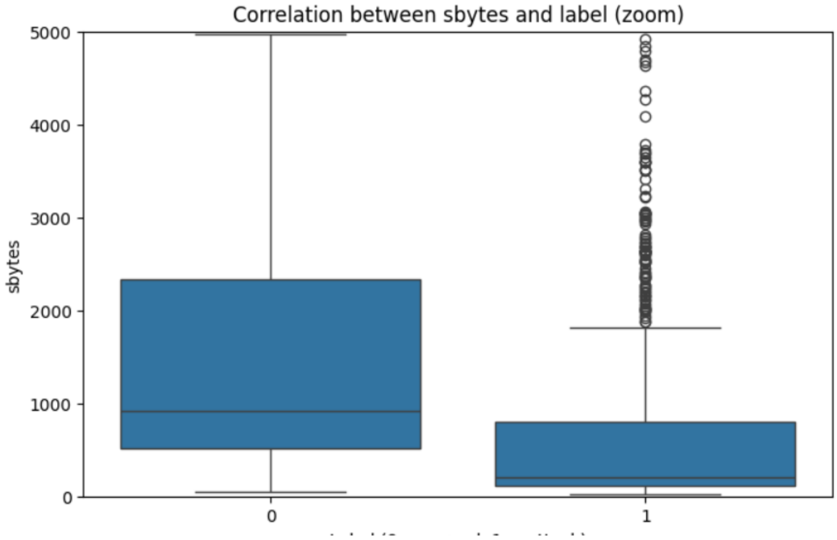


Figure 2: Correlation between sbytes and label (zoomed in)

The boxplot shows that normal traffic (label = 0) usually sends more bytes than attacks. In contrast, attacks (label = 1) often have lower `sbytes` values, indicating smaller and faster connections. However, some attacks also produce a lot of bytes, which means some malicious flows can send large amounts of data.

3.3 Attack Rate per Service and per State

To better understand how attacks are distributed within the dataset, we analysed two categorical rows: the `service`, which indicates the protocol involved in the connection, and the `state` field, which describes the connection status or the network response

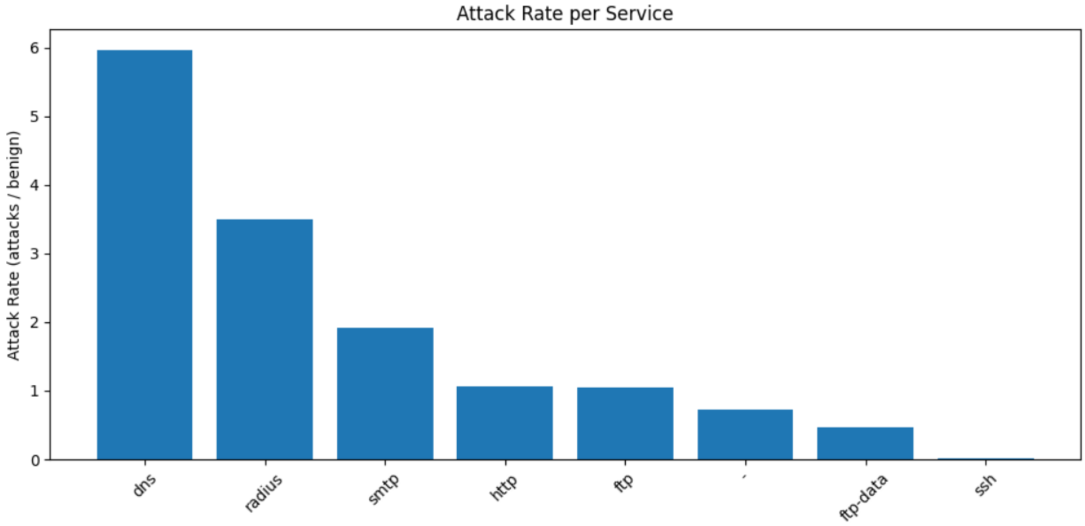


Figure 3: Attack rate per service

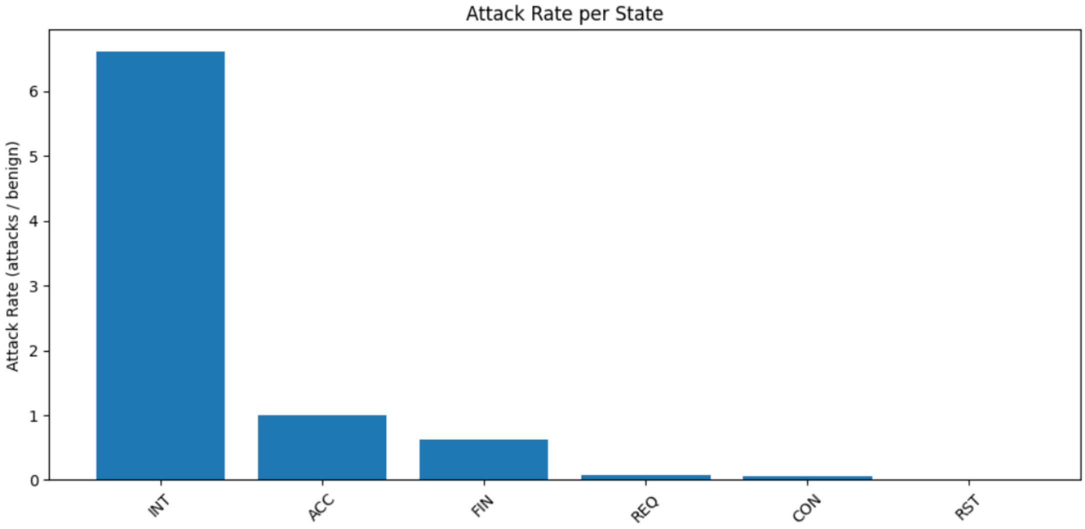


Figure 4: Attack rate per protocol state

The results show us how attacks are distributed and also show patterns. Services such as `dns` and `radius` have the biggest attack numbers, meaning that DNS and authentication services are frequently targeted by malicious activity, meanwhile `ssh` and `ftp-data` have low attack numbers.

A similar scheme appears in the connection states: the `INT` (internal) state has a high attack number compared to other states such as `ACC`, `FIN`, or `REQ`.

3.4 Correlation Matrix

To generalize our analysis to all features, we compute the correlation matrix.

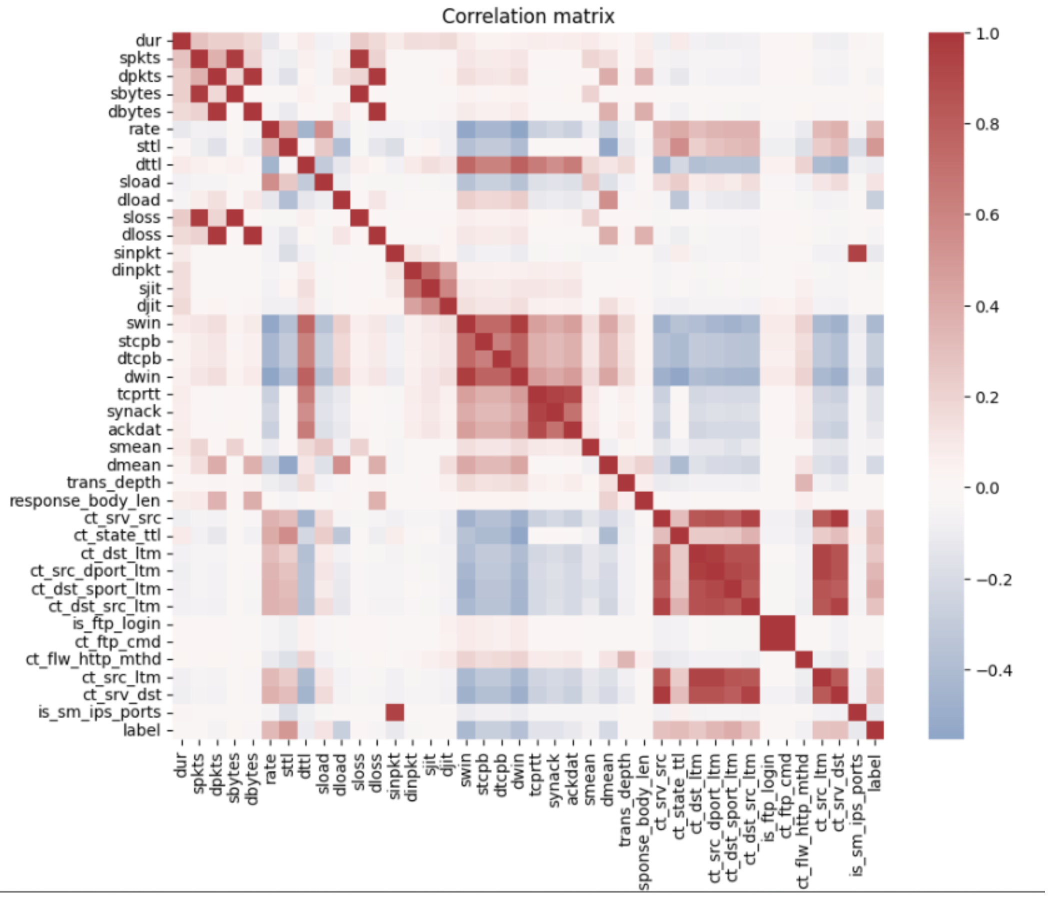


Figure 5: Correlation matrix of all numerical features

In this matrix, lighter colors show stronger correlations (values close to 1), while darker areas indicate weak or no correlation (values close to 0). We can especially notice:

- strong positive correlation between `spkts`, `sbytes` , and `sloss`. This is intuitive: more packets mean more data, and therefore more packet loss;
- the same effect appears for `dpkts`, `dbytes` , and `dloss`.

This shows that not all features have the same importance. We will use this insight later during prepossessing and PCA.

3.5 Protocol Analysis

We now separate the label from the rest of the data to see how some variables, such as the transmission protocol, influence the probability of attack.

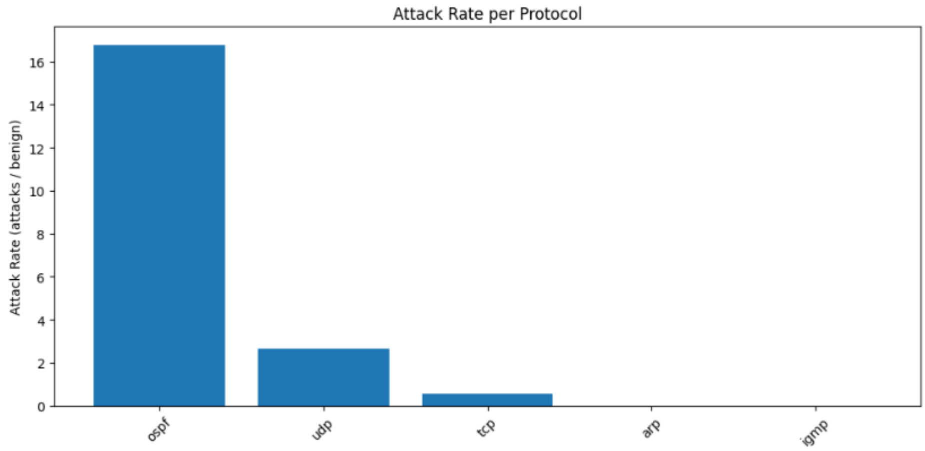


Figure 6: Attack rate per protocol

The plot suggests that `ospf`, `udp` and `tcp` can be considered as more exposed or more subject to attacks than other protocols. Less used protocols can be ignored or grouped during prepossessing.

3.6 Label and Attack Category Distribution

The dataset is relatively balanced:

- Normal traffic: 37 000 (44.94%)
- Malicious traffic: 45 332 (55.06%)

So we do not need resampling or stratified k-fold cross-validation for the binary label.

The attack category (`attack_cat`) is more heterogeneous:

- Generic: 18 871
- Exploits: 11 132
- Fuzzers: 6 062
- DoS: 4 089
- Reconnaissance: 3 496
- Analysis: 677
- Backdoor: 583
- Shellcode: 378
- Worms: 44

Generic attacks are much more frequent than, for example, Backdoor or Worms.

4 Preprocessing

The preprocessing step prepares the raw dataset for machine learning. It includes:

- removing unexploitable values;
- removing duplicates;
- separating features and target (label);
- encoding categorical variables (protocol, service, state, etc.);
- scaling numerical variables;
- splitting the data into training and testing sets.

4.1 Encoding Text Columns

Some columns contain text, such as `proto`, `service`, and `state`. Machine learning models cannot use text directly, so we replace them with numbers:

- we create custom risk scores for protocol, service, and state using graphs from analysis and create new columns such as score risk.
- we map each value to its score;
- we remove the original text columns afterwards.

4.2 Adjusting Numerical Columns

The `sbytes` and `dbytes` columns have very large values and many outliers. To make them easier to work with, we apply a log transformation and keep only the log versions.

4.3 Removing Useless or Redundant Columns

We drop columns that :

- do not bring useful information;
- are too specific, such as isftplogin;
- or repeat information already contained in other variables, such as spkts or sloss.

We also drop `attack_cat` so that the algorithms only use the binary label and do not overfit on this very informative feature.

Finally, we standardize all numerical columns, which makes all values comparable and avoids giving too much weight to large-scale variables.

We then split the data into:

- 70% for training,
- 30% for testing.

5 Principal Component Analysis (PCA)

PCA is used to reduce the dimensionality of the dataset while keeping most of the information. Since the dataset contains many numerical and categorical features, some of which are redundant or weakly informative, PCA helps simplify the representation space and reduce noise.

After fitting PCA on the preprocessed training data, we selected the number of components required to explain at least 95% of the total variance. This led to a reduced representation of the dataset using **23 principal components**.

To better understand which variables contribute most to the variance captured by PCA, we examined the feature importances derived from the PCA loadings. The most influential features mainly correspond to traffic volume indicators and low-level communication indicators.. The **top 10 most important features** are:

- `dload` (importance = 0.1629)
- `rate` (0.1567)
- `dmean` (0.1563)
- `smean` (0.1437)
- `ct_state_ttl` (0.1424)
- `sload` (0.1420)
- `ct_flw_http_mthd` (0.1411)
- `stcpb` (0.1397)
- `sttl` (0.1396)
- `dur` (0.1331)

These features are associated with packet flow intensity, time-to-live, connection behavior, and byte-level statistics, which seem to be good at indicating abnormal behaviours.

In contrast, several encoded categorical features have very little contributin : The **10 least important features** include:

- `proto_smp` (importance = 0.000077)
- `proto_merit-inp` (0.000072)
- `proto_ipv6-opts` (0.000068)
- `proto_sdrp` (0.000067)
- `proto_idrp` (0.000065)
- `proto_dgp` (0.000059)
- `service_radius` (0.000058)
- `service_irc` (0.000025)
- `state_CLO` (0.000018)
- `state_ACC` (0.000007)

These variables contribute very little to the variance and can be considered non-informative for the classification.

6 Presentation of the Models

To address this classification problem, we tested several well-known machine learning algorithms, each representing a different family of models.

Logistic Regression

A simple linear classifier used as a baseline. It tries to separate the two classes with a straight line. Even though it is basic, it performed reasonably on our dataset and provided a useful reference.

Decision Tree

A model that learns decision rules from the data and organizes them into a tree structure. It is easy to understand but can overfit if the data is not preprocessed well. It achieved good results on our dataset.

Support Vector Machine (SVM)

A strong classifier that finds the best margin between classes. SVM works very well with high-dimensional data and is often used in security applications.

Random Forest

An ensemble of many decision trees trained on different subsets of the data. It reduces overfitting and captures complex patterns. It performed extremely well and was one of the top models among the non-ensemble techniques we tested first.

Gradient Boosting

A boosting method that builds a series of weak learners, each one improving the previous one. It is powerful but more sensitive to noise. It still showed high performance and remained a solid model.

Bagging (Bootstrap Aggregating)

Bagging trains many models on different bootstrap samples of the data and combines their predictions. In our tests, Bagging performed very well, with strong accuracy and F1-score, and reduced overfitting. Its results were close to Random Forest.

AdaBoost

AdaBoost trains models one after another, giving more weight to misclassified samples. In our results, AdaBoost performed correctly but slightly lower than Bagging and Random Forest, which is expected since it can be more sensitive to noisy data.

Stacking Classifier

Stacking combines several different models (Decision Tree, SVM, Random Forest) and trains a final model to learn how to best mix their predictions. In our dataset, Stacking achieved the best performance among the ensemble methods, with very strong precision and recall.

7 How We Addressed the Obstacles

At the beginning of the project, we planned to work with the EMBER dataset, well-known for malware detection. EMBER contains static features extracted from PE (Portable Executable) files and is used for machine learning to train models that classify binaries as malicious or benign.

You can check out what we did on our Github in OldProject or by clicking on the following link : Ember-project-Github

However, the dataset is extremely large (over one million samples) and requires heavy preprocessing, feature extraction, and memory management.

We managed to do a huge part of the project on this dataset, however training Ensemble methods models was simply impossible as it ran our computers out of memory.

Very quickly, we encountered other issues : The dataset size exceeded the capabilities of our computers, making it impossible to load, preprocess, or train models. We also faced compatibility issues between Python versions, package conflicts on Windows, and execution fails caused by the volume of the dataset.

Because of these issues, we decided that finishing our project with ember was simply not possible and chose to switch to a lighter, more manageable dataset: UNSW-NB15. You can see what we did on the Ember dataset on the second GitHub that we sent.

During the new project, we faced several challenges related to data quality and model performance.

High Dimensionality

The UNSW-NB15 dataset contains many numerical and categorical features, some of which are redundant or noisy. To address this, we used PCA to reduce the number of dimensions while keeping 95% of the total variance. This reduced the computational cost, simplified the modeling phase, and focused the learning process on the most relevant information.

Risk of Overfitting

Complex models such as Random Forest or Gradient Boosting can easily overfit when the feature space is large or contains irrelevant attributes. By applying PCA and removing low-importance columns, we reduced the risk of overfitting and improved generalization. We also validated our results using a clean train/test split to ensure that performance was stable and not the result of overfitting.

8 Comparison of Model Results Without Ensemble Methods

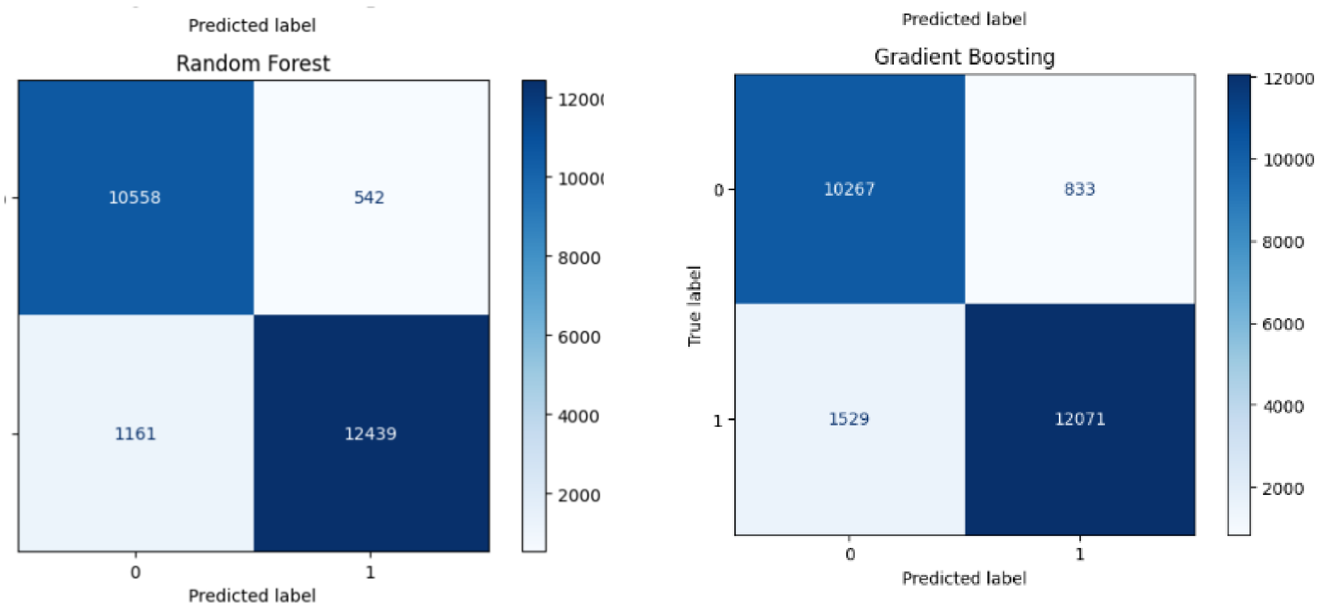
After training all classical models on the PCA-transformed data, we compared them using Accuracy, Precision, Recall, and F1-score.

Table 1: Performance of non-ensemble models

Model	Accuracy	Precision	Recall	F1
Logistic Regression	0.835587	0.834256	0.875294	0.854283
Decision Tree	0.907166	0.917017	0.914118	0.915565
SVM	0.907895	0.959507	0.869412	0.912240
Random Forest	**0.931053**	**0.958247**	**0.914632**	**0.935932**
Gradient Boosting	0.904372	0.935446	0.887574	0.910881

For non-ensemble methods, **Random Forest** is the best model on this dataset, with the highest overall performance.

Confusion Matrices and ROC Curve



(a) Random Forest (b) Gradient Boosting

Figure 7: Confusion matrices for classical models

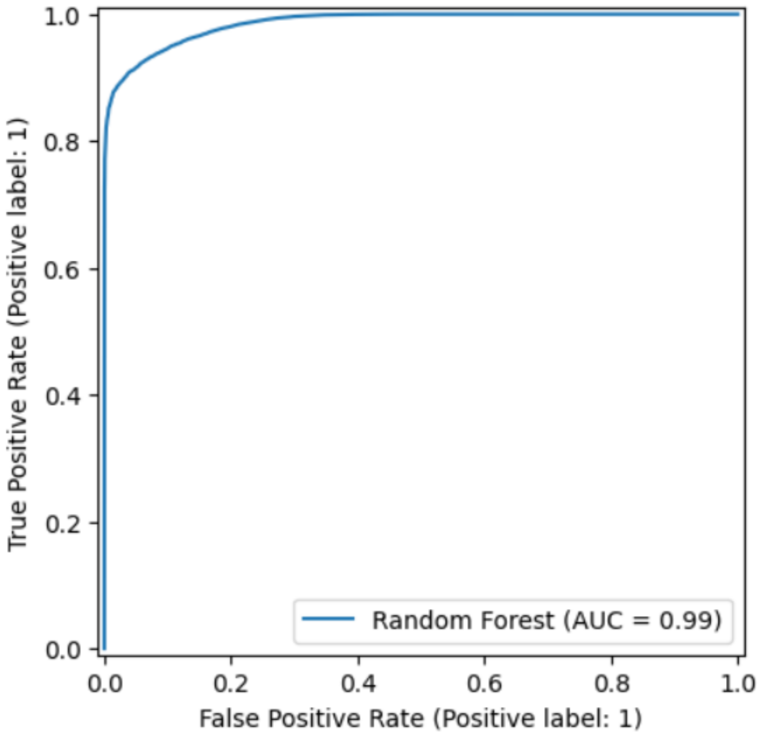


Figure 8: ROC curve for Random Forest (AUC = 0.99)

9 Comparison of Ensemble Methods

Using the PCA-transformed data, we also tested three additional ensemble models: Bagging, AdaBoost, and Stacking. These models combine several classifiers to improve stability and accuracy.

Table 2: Performance of ensemble methods

Model	Accuracy	Precision	Recall	F1
Bagging	0.9296	0.9543	0.9160	0.9348
AdaBoost	0.8987	0.9209	0.8926	0.9066
Stacking	**0.9310**	**0.9489**	**0.9246**	**0.9365**

Overall, Stacking gives the best results, slightly outperforming Bagging and AdaBoost. It also reaches performance close to Random Forest and even improves recall and F1-score. This shows that combining several models can give a more balanced and more stable classifier.

Confusion Matrices

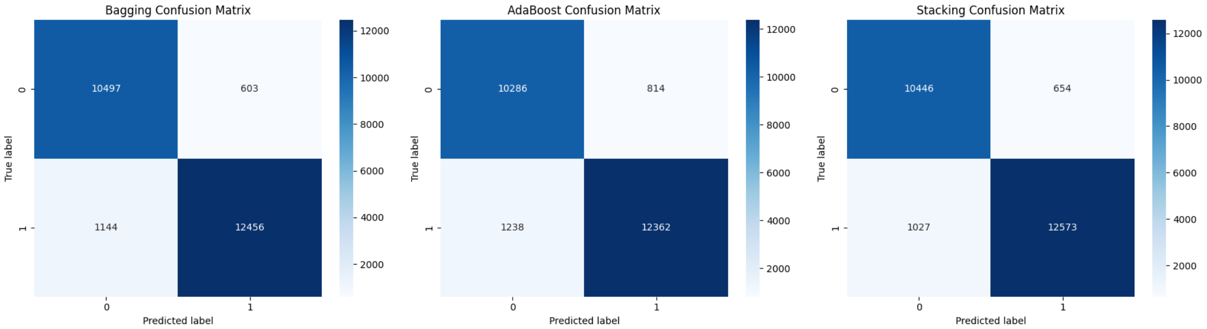
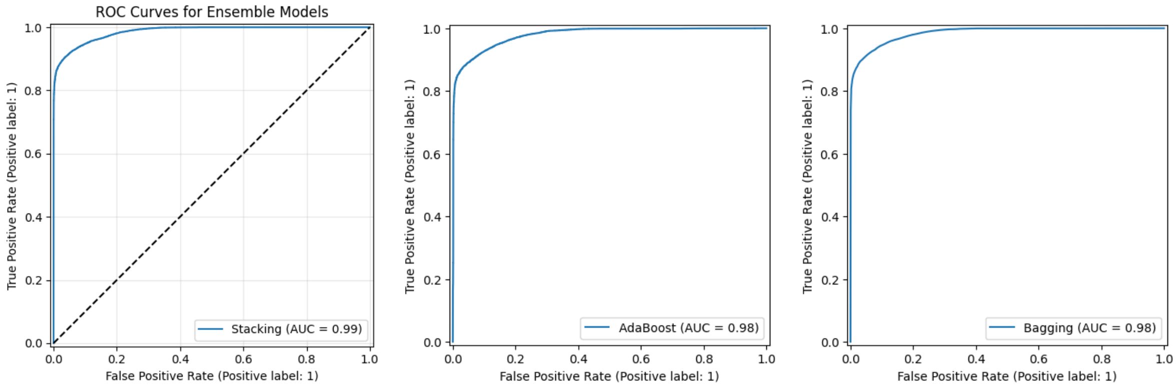


Figure 9: Ensemble methods Confusion matrixes

Figure 10: Confusion matrices for the ensemble models

The confusion matrices show that Stacking makes fewer classification errors than the other two methods, further confirming its efficiency.
This is particularly true on the attack samples as we can see, meaning that this model is actually performing really well in exactly what we expect from it.

ROC Curves



(a) Stacking (b) AdaBoost (c) Bagging

Figure 11: ROC curves for ensemble models

All three models reach very high AUC values (around 0.98–0.99), showing strong ability to separate normal and attack traffic. Stacking again has the best curve, confirming its solid performance.

10 Conclusion on the Business Case

The goal of this project was to build an automatic system that can detect intrusions based on network activity.

Using the UNSW-NB15 dataset, we explored and cleaned the data, removed unnecessary data, encoded text values, reduced the number of features with PCA, trained several machine learning models, and compared their performance.

The results show that machine learning can clearly separate normal traffic from malicious traffic.

Among the models we tested, **Random Forest** and the **Stacking** method gave the best overall performance, with strong accuracy, precision, and recall. **SVM** also performed very well, especially in terms of precision, but had lower recall compared to these two models.

These results confirm that intrusion detection can help companies or organizations react faster and more reliably to cyberattacks. Our study also shows that using a clean preprocessing and different models models greatly improves detection quality and makes the system more useful and adapted to detecting malicious network traffic.

11 Scientific References & Sources

Main scientific papers associated with the dataset:

- Moustafa, N., & Slay, J. (2015). *UNSW-NB15: a comprehensive data set for network intrusion detection systems.*
- Moustafa, N., & Slay, J. (2016). *The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 dataset.*
- Koroniotis, N., Moustafa, N., et al. (2017). *Towards Developing Network Forensic Mechanisms for Botnet Activities in IoT.*

Dataset source: **Kaggle – UNSW-NB15 Network Intrusion Detection**.