ARISTOTLE UNIIVERSITY OF THESSALONIKI

FACULTY OF NATURAL SCIENCES

DEPARTMENT OF PHYSICS

P.P.S. (MS) IN ELECTRONIC PHYSICS (RADIOELECTROLOGY)

Master Thesis

A supervised machine learning framework for anomaly-based intrusion detection

Anastasia Chaloulakou

(10024)

Supervisors

K. Siozios, *Associate Professor, Dpt. Of Physics, AUTH*

M. Dasygenis, *Assistant Professor, Dpt. Of Electronic and Computer Engineering, UoWM*

Thessaloniki, October 2022

Acknowledgments

I would like to thank my supervisors, mr. Kostas Siozios and mr. Minas Dasygenis, for their guidance and support throughout the whole process of creating this thesis.

I also want to thank Dimitris Tsiktsiris, for his thoughtful advice and help.

The IT department of AUTH, and especially mr. Georgios Pallas, for their help in gathering and preparing the traffic from the university gateway, even though that part of the project failed to launch in the end.

Lastly, I would like to express my gratitude to my family and friends, for the years of constant support, help and encouragement that they generously provided me with throughout my undergraduate and postgraduate studies, and for their patience while I was ranting about things they didn't understand.

This thesis was partially supported by the Hellenic Petroleum Company and the Special Account for Research Funds, through the Scholarship for Academic Excellence during the academic year 2020-2021

Abstract

With the rapid development of networks and Internet services, network security has gained increased momentum in the past few years. Consequently, Intrusion Detection Systems (IDS) must adapt to the increased need for a sufficient first line of defence against the ever-evolving threats landscape. By utilizing deep and machine learning techniques, IDSs have been focusing on anomaly detection, but there are still challenges in detecting attacks, especially rare or novel ones, due to the unavailability and imbalance of data. Furthermore, there are many attacks that have not yet been discovered and analysed, and they continue to evolve every day. In this thesis, the NSL-KDD dataset, one of the most popular benchmark datasets available, is analysed and used in five common supervised learning classification algorithms. Despite the simplicity of the models, they show a good performance that is almost on par with state-of-the-art deep learning and unsupervised models, thus providing us with a coherent review of how machine learning is used for anomaly detection and where it can go from there.

Περίληψη

Με την ραγδαία ανάπτυξη των δικτύων και των υπηρεσιών μέσω Διαδικτύου, η ασφάλεια έχει αποκτήσει μεγάλη ώθηση τα τελευταία χρόνια. Συνεπώς, τα Συστήματα Ανίχνευσης Εισβολών (ΣΑΕ) πρέπει να προσαρμοστούν στην αυξημένη ανάγκη για μια επαρκή πρώτη γραμμή άμυνας του δικτύου ενάντια στις συνεχώς εξελισσόμενες απειλές. Με τη χρήση τεχνικών βαθιάς και μηχανικής μάθησης, τα ΣΑΕ έχουν επικεντρωθεί σε λειτουργίες ανίχνευσης ανωμαλιών, όμως υπάρχουν ακόμα προκλήσεις στην αναγνώριση επιθέσεων, ειδικά όταν είναι πιο σπάνιες ή καινούριες, λόγω της μη διαθεσιμότητας δεδομένων, και την άνιση κατανομή των δεδομένων. Επιπλέον, υπάρχουν πολλές επιθέσεις που ακόμα δεν έχουν ανακαλυφθεί και αναλυθεί, οι οποίες εξελίσσονται καθημερινά. Σε αυτή την εργασία το πακέτο δεδομένων NSL-KDD, ένα από τα πιο διαδεδομένα διαθέσιμα πακέτα, αναλύεται και έπειτα χρησιμοποιείται για την αξιολόγηση πέντε μοντέλων ταξινόμησης επιβλεπόμενης μηχανικής μάθησης. Παρόλη την απλότητα των μοντέλων, καταφέρνουν να φτάσουν σε καλή απόδοση, συγκρίσιμη με state-of-the-art μεθόδων βαθιάς και μη επιβλεπόμενης μάθησης, προσφέροντας έτσι μια συνοπτική συγκεντρωτική επισκόπηση του πώς χρησιμοποιείται η μηχανική μάθηση στην ανίχνευση ανωμαλιών, και πώς μπορεί να εξελιχθεί ακόμα περισσότερο.

Εκτεταμένη περίληψη

Οι περισσότερες διαδικασίες και υπηρεσίες σήμερα γίνονται μέσω του Διαδικτύου. Η δικτύωση έχει αναπτυχθεί πολύ τα τελευταία χρόνια, και θα συνεχίσει να εξελίσσεται, χάρη στην ευρεία εφαρμογή του 5G δικτύου και την έρευνα που ήδη γίνεται στο 6G. Λόγω του σημαντικού ρόλου που παίζουν τα δίκτυα και το διαδίκτυο στην κοινωνία μας, η κυβερνοασφάλεια έχει γίνει ζωτικής σημασίας για την προστασία των δεδομένων και των συσκευών μας. Τα Συστήματα Ανίχνευσης Εισβολών (ΣΑΕ) αποτελούν σημαντικό κομμάτι τόσο της ασφάλειας, όσο και της ίδιας της δομής του δικτύου, καθώς μπορούν να ανιχνεύσουν και να αποτρέψουν κακόβουλα προγράμματα και χρήστες από το να παραβιάσουν το δίκτυο, και να σταματήσουν διάφορα είδη επιθέσεων προτού αποδειχθούν επικίνδυνες. Με τη ραγδαία εξέλιξη της μηχανικής μάθησης και της τεχνητής νοημοσύνης, η δομή των ΣΑΕ αλλάζει από τεχνικές βασισμένες σε «υπογραφές», δηλαδή που αναγνωρίζουν συγκεκριμένα μοτίβα γνωστών επιθέσεων, σε πιο αφηρημένες/γενικευμένες μορφές λειτουργίας βασισμένης σε αναγνώριση ανωμαλιών, οι οποίες ταξινομούν την κίνηση ως φυσιολογική ή επικίνδυνη.

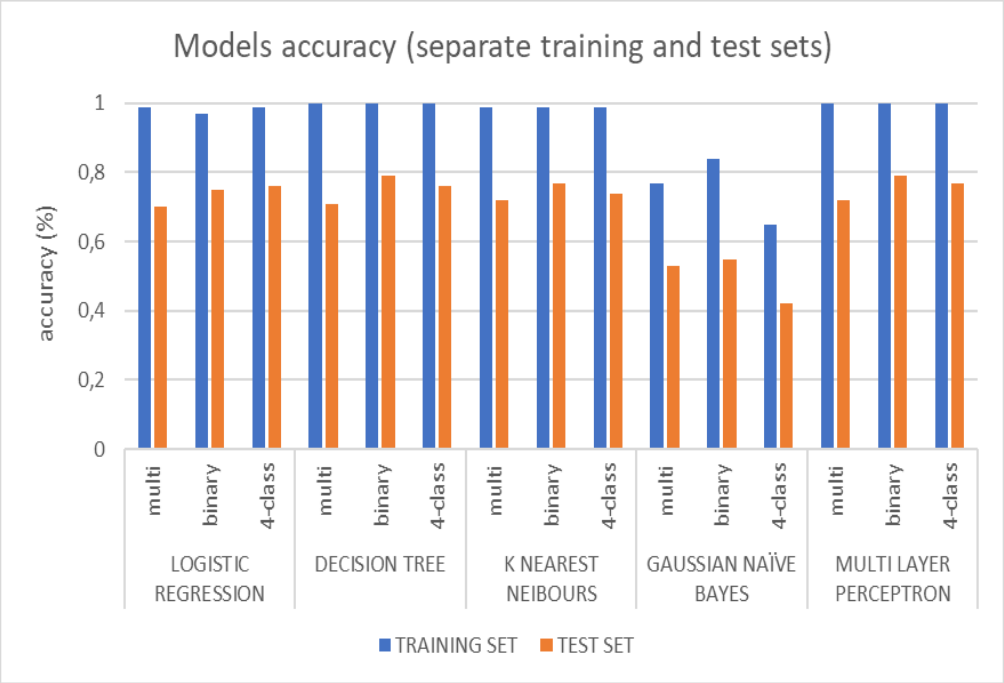
Οι ανωμαλίες ενός δικτύου μπορεί να προέρχονται από κακόβουλες δραστηριότητες που εκμεταλλεύονται υπηρεσίες δικτύου, υπερφόρτωση από δεδομένα, δυσλειτουργικές συσκευές και υπονόμευση διαφόρων παραμέτρων του δικτύου [1], και μπορεί να σχετίζονται είτε με τις επιδόσεις του (π.χ. υπερχείλιση δεδομένων λόγω κάποιας υπολειτουργικής μονάδας του δικτύου) είτε με την ασφάλεια (π.χ. εκ προθέσεως υπερχείλιση του δικτύου ώστε οι χρήστες να μην έχουν πρόσβαση στις υπηρεσίες). Τα ΣΑΕ μπορούν να ανιχνεύσουν οποιαδήποτε απόκλιση από την φυσιολογική συμπεριφορά, για αυτό είναι καλύτερα από τα κλασσικά συστήματα υπογραφών στο να ανιχνεύουν καινούριες ή άγνωστες επιθέσεις, αυτό όμως έρχεται με το κόστος ότι δίνουν περισσότερες λανθάνουσες ειδοποιήσεις.

Το NSL-KDD πακέτο δεδομένων είναι ένα από τα πιο συχνά χρησιμοποιούμενα πακέτα δεδομένων δικτύου, από όταν δημιουργήθηκε το 2009 [2][3][4]. Συνεχίζει μέχρι σήμερα να χρησιμοποιείται στην έρευνα σαν benchmark για μοντέλα ανίχνευσης ανωμαλιών στα δίκτυα, όπως στα παραπάνω άρθρα. Για αυτό, επρόκειτο για ένα εξαιρετικό πακέτο δεδομένων για τη σύγκριση των διαφόρων μοντέλων που δοκιμάστηκαν σε αυτήν την εργασία, για μια αξιόπιστη πηγή διαφόρων ειδών επιθέσεων και επιπέδων δυσκολίας ανίχνευσης, τόσο στο πακέτο της εκπαίδευσης όσο και του ελέγχου των μοντέλων. Επιπρόσθετα, οι διαφορές μεταξύ των δύο αυτών πακέτων παρείχαν μια πιο ρεαλιστική εικόνα της δυνατότητας των μοντέλων να ταξινομήσουν σωστά την κίνηση του δικτύου.

Σε αυτή την εργασία, σκοπός είναι να χρησιμοποιηθεί το NSL-KDD για τη σύγκριση πέντε από τις πιο διαδεδομένες μεθόδους μηχανικής μάθησης σε εφαρμογές ταξινόμησης, οι οποίες είναι: logistic regression, k nearest neighbours, decision tree, Gaussian Naive Bayes και multilayer perceptron. Έτσι, στην ***ενότητα 2*** βρίσκεται μια συνοπτική εισαγωγή στη μηχανική μάθηση για ανίχνευση ανωμαλιών, όπως και συναφής έρευνα που γίνεται τα τελευταία χρόνια. Επίσης, αναφέρονται τα προτερήματα του NSL-KDD. Η ***ενότητα 3*** παρέχει πληροφορίες για τους πέντε αλγορίθμους που χρησιμοποιήθηκαν στην εργασία. Στην ***ενότητα 4***, μετά τη δημιουργία τριών εκφάνσεων του πακέτου δεδομένων, έτσι ώστε να συγκριθούν τα

διαφορετικά σενάρια ταξινόμησης (όλες οι επιθέσεις, φυσιολογική/επικίνδυνη κίνηση, 4 κατηγορίες επιθέσεων), το NSL-KDD αναλύεται και στη συνέχεια προετοιμάζεται για να εισαχθεί στα μοντέλα μηχανικής μάθησης. Τέλος, στην ***ενότητα 5*** τα μοντέλα βελτιστοποιούνται, αξιολογούνται και τα αποτελέσματα συγκρίνονται με αυτά της σχετικής έρευνας, ενώ στην ***ενότητα 6*** συζητούνται τα προβλήματα και οι περιορισμοί τόσο αυτού του πειράματος, όσο και της ανίχνευσης ανωμαλιών συνολικά, όπως και μελλοντική δουλειά πάνω στο αντικείμενο.

Τα αποτελέσματα της έρευνας που έγινε παρουσιάζονται στην *Εικόνα 1*, όπου μπορούν άμεσα να συγκριθούν οι επιδόσεις του κάθε μοντέλου. Παρατηρούμε ότι έχουμε ακρίβεια $70 - 79\%$, με εξαίρεση τον αλγόριθμο Gaussian Naive Bayes, ο οποίος λειτουργεί με την προϋπόθεση ότι δεν έχουν καθόλου εξάρτηση η μία μεταβλητή του dataset από την άλλη, πράγμα που στην περίπτωσή μας δεν ισχύει καθόλου.



Εικόνα 1: *συνοπτικό διάγραμμα της επίδοσης όλων των μοντέλων ταξινόμησης, σε όλα τα σενάρια κατηγοριοποίησης της κίνησης δικτύου, με χρήση του KDDTrain+ για την εκπαίδευση των μοντέλων και του KDDTest+ για τον έλεγχο/επαλήθευση*

Η ακρίβεια αυτή των μοντέλων είναι συγκρίσιμη με την ακρίβεια που πετυχαίνουν μοντέλα συγγενούς έρευνας που γίνονται τα τελευταία χρόνια *(βλ. ενότητα: 5.2. Evaluation and results compared to relevant research)* παρόλο που στις περισσότερες περιπτώσεις εκείνων των προγραμμάτων χρησιμοποιούνται πολύ πιο σύνθετα, μεγάλα και βαθιά μοντέλα.

Πιο συγκεκριμένα, στην πιο πρόσφατη παρόμοιας δομής έρευνα που αναλύθηκε ([26][27][28][29]), χρησιμοποιούνται μοντέλα που περιέχουν καινοτόμες τεχνικές βαθιάς μάθησης, μεταξύ άλλων convolutional και contractive autoencoders (μέθοδοι αυτο-επιβλεπόμενης και μη επιβλεπόμενης μάθησης αντίστοιχα), Deep Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long-Short Term Memory (LSTM),

μηχανισμοί προσοχής, κ.α.. Εκεί ακόμα επιτυγχάνεται ακρίβεια $75 - 89\%$ ανάλογα με την περιπλοκότητα του εκάστοτε μηχανισμού και την εξαγωγή χαρακτηριστικών που έχει υποστεί το dataset.

Μιας και πρόκειται για τόσο διαφορετικού βάθους και καινοτομίας μεθόδους, με λεπτομερή βελτιστοποίηση σε πολλά στάδια, η δική μας έρευνα, που έγινε με πολύ πιο απλά μέσα και τεχνικές και όμως βρίσκεται σε απόδοση κοντινή με το state of the art, μπορεί να αποτελέσει μια αξιόπιστη μελέτη των βασικών αυτών μεθόδων ταξινόμησης που χρησιμοποιήθηκαν, προς σύγκριση μεταξύ των αλγορίθμων, των διαφορετικών κατηγοριών κίνησης, και ανάλυση των μηχανισμών λειτουργίας τους.

Η ανίχνευση ανωμαλιών και γενικότερα η κυβερνοασφάλεια αντιμετωπίζει ακόμα πολλές προκλήσεις. Αναφορικά, μερικές από αυτές είναι:

- Η ραγδαία ανάπτυξη των δικτύων σήμερα, η οποία οδηγεί σε αύξηση των καινούριων και άγνωστων επιθέσεων που εκμεταλλεύονται καινούρια κενά και υπηρεσίες.
- Η όλο και μεγαλύτερη εξάρτηση της κοινωνίας μας από το Διαδίκτυο, στο οποίο κάθε χρόνο παράγονται και διακινούνται πολύ περισσότερα δεδομένα, ήδη δύσκολα επεξεργάσιμα με τα σημερινά μέσα.
- Το Διαδίκτυο των Αντικειμένων (Internet of Things – IoT), λόγω του οποίου συσκευές όλο και χαμηλότερου επιπέδου, άρα και με όλο και λιγότερες υπολογιστικές δυνατότητες, συνδέονται μεταξύ τους, αφήνοντάς μας εκτεθειμένους σε κενά ασφαλείας που θα μπορούσαν να έχουν επιπτώσεις ακόμα και στη σωματική μας υγεία, πέρα από την ασφάλεια των δεδομένων μας.
- Η μη διαθεσιμότητα ανοιχτών πακέτων δεδομένων κίνησης δικτύου, ιδιαίτερα πρόσφατων, που θα περιέχουν πιο καινούριες επιθέσεις, λόγω ιδιωτικότητας, ανταγωνισμού των παροχών δικτύου, που θα μπορούσαν να ανανεώσουν τον χώρο της έρευνας.
- Η ανεπάρκεια της μη επιβλεπόμενης μάθησης, παρόλο που ενδείκνυται για την ανίχνευση ανωμαλιών, καθώς από unlabelled δεδομένα δεν μπορούν οι ερευνητές να ξέρουν την πραγματική απόδοση της, ενώ ταυτόχρονα η δημιουργία labels (ετικετών) στα πακέτα αποτελεί μια ιδιαιτέρως δύσκολη και χρονοβόρα διαδικασία.

Συνοψίζοντας, παρόλες τις προκλήσεις και τους περιορισμούς που αντιμετωπίζει η ανίχνευση ανωμαλιών στο χώρο της ασφάλειας δικτύων, η έρευνα αναπτύσσεται μαζί με τον τομέα της μηχανικής μάθησης και τεχνητής νοημοσύνης, ακολουθώντας τις πιο καινοτόμες μεθόδους. Η μη επιβλεπόμενη μάθηση μπορεί γίνει πιο χρήσιμη στον πραγματικό κόσμο, μιας και τα δεδομένα που χρησιμοποιεί δε χρειάζονται labelling, και μπορεί να ανιχνεύει άγνωστες και καινούριες επιθέσεις. Παρά την τάση που έχει ξεκινήσει τα τελευταία δύο χρόνια να διερευνηθούν οι τεχνικές μη επιβλεπόμενης μάθησης, η επιβλεπόμενη μάθηση παραμένει ακόμα ο κύριος τρόπος που μελετάται η ασφάλεια των δικτύων, και πέντε από τους πιο βασικούς αλγορίθμους μηχανικής μάθησης σε προβλήματα ταξινόμησης, αν και κάπως παρωχημένοι πια, μελετήθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας, δίνοντας αποτελέσματα σε πολύ ικανοποιητικό επίπεδο.

Table of Contents

Figures index:

Tables index:

Equations index:

1. Introduction

In today's world, most processes and services of everyday life pass through the Internet. Networking has advanced greatly in the past few years, and will continue to do so, with the vast implementation of 5G and 6G that is already being tested and researched. Because of the important role that networks and the Internet play in our society, cyber security has become vital for the protection of our data and devices. Intrusion Detection Systems (IDS) are an important part of cyber security and of the network's infrastructure, as they can detect and prevent the malicious programs and users from breaching the network and stop various kinds of attacks before they pose a danger. With the rapid growth of machine learning and artificial intelligence (AI), IDSs have shifted from signature-based techniques, that work by recognising specific patterns in mostly known attacks, to more abstract anomaly-based detection, which classifies traffic as normal (safe) and abnormal (dangerous).

Anomalies in a network can be caused by malicious activities that take advantage of network services, overload of traffic, malfunctioning devices and compromising various network parameters [1], and can be performance-related (e.g., traffic flooding because of a malfunctioning node) or security-related (e.g., intentional flooding of the network resources so that legitimate users cannot access the services). Anomaly detection systems can detect any kind of deviation from the normal behaviour, so they are better than more classical signature-based systems at catching novel and unknown attacks; however, it comes at the cost of raising more false alarms.

The NSL-KDD dataset is one of the most commonly used network traffic sets ever since its creation in 2009 [2][3][4]. It is still used in research as a benchmark for network traffic classification models, like all of the papers cited here. Thus, it provided an excellent dataset for the comparison of the different machine learning models tested, for a reliable source of different types of attack labels and high difficulty level of attacks in both the training and the test sets. In addition, the differences between the two subsets provided for a good real-world test of the models' abilities to classify correctly.

In this thesis, our objective is to use the NSL-KDD dataset to compare five of the most commonly used supervised learning classification models, which are: logistic regression, k-nearest neighbours, decision tree, Gaussian Naïve Bayes, and the multi-layer perceptron. For this purpose, ***section 2*** provides a brief introduction to machine learning techniques for anomaly detection, as well as relevant research that has been carried out in the past couple years; it also discusses the advantages of the NSL-KDD dataset. ***Section 3*** gives more information on the five algorithms that are used for our experiment. In ***section 4***, after creating three instances of the dataset, in order to compare the different classification scenarios (multiclass, binary, and 4-class classification), the NSL-KDD dataset is initially analysed and pre-processed, and subsequently fed into the different models and optimised for best accuracy scores. Lastly, in ***section 5***, the models are evaluated, and the results of this research discussed, and ***section 6*** focuses on the problems that anomaly detection is still facing, as well as future work and research on the topic.

2. An overview of machine learning and anomaly detection research

Machine learning has been one of the most rapidly advancing technologies for years now, and continues to grow even more, with the advancement of computational power, artificial intelligence (AI) and Internet of Things (IoT). In the domain of cyber security, machine learning has greatly influenced the way networks are protected, which is something crucial in the era of the Internet Services.

Intrusion Detection Systems (IDS) are now capable of recognising unknown attacks that try to penetrate the network, by scanning the traffic for anomalies. Anomalies in the network are all instances in the data that do not conform to the behaviour exhibited by normal traffic [1]. There don't necessarily have to be malicious attacks, as *performance-related* anomalies also occur in the network (traffic overload, malfunctioning devices, etc). However, anomalies in data can translate to significant and often critical problems with the information passed through the network. In network security, the anomalies researchers and the relevant systems are looking for are *security-based*, which means that they stem from malicious actions against the network. These intrusions to the network aim to compromise the confidentiality, integrity or availability of a system or service, by bypassing the security mechanisms built in the network's infrastructure. As a result, security experts use IDS in order to protect the network from outside threats.

An IDS is a software and/or hardware system that monitors the events occurring in a network and analyses them for signs of intrusion by unwanted traffic (malicious activity). IDSs can be *signature-based*, that can only detect known attacks, and need constant updating from the vendors in order to keep up with the rapidly growing new malware, or they can be *anomaly-based*, which can capture any deviations from normal behaviour, and are better at recognising attacks that were previously unknown. However, they generate a large number of false alarms, due to the limitations of their capabilities and training.

Anomaly detection IDSs rely heavily on machine learning, since their function is to classify data based on what is considered normal traffic and deviations from it. The fact that they require training is the reason they have limited capabilities still. There are four machine learning model categories that can be applied to anomaly detection: a) supervised, b) semi-supervised, c) unsupervised and d) hybrid training models. In supervised training, the IDS model trains on labelled data, from a dataset that contains both normal and malicious traffic and any unseen instance is compared to the model to determine which class it belongs to. In semi-supervised models, training data contains only normal data instances, thus it cannot differentiate between attack classes when it encounters malicious traffic, only normal and abnormal events. With unsupervised training methods, the model doesn't require any training data, which would make it the most widely applicable way, but the unlabelled nature of the data makes them less useful and quantifiable in their performance. Naturally, the hybrid approach combines features of all the aforementioned methods, to create the optimal result for large scale applications.

In this thesis, the models that were developed and compared are all supervised learning algorithms, thus a more thorough explanation of the way this kind of machine learning works in our context will be given in the next section.

2.1. Supervised machine learning

Supervised anomaly detection systems are based on prior knowledge that they acquired during training. They build a predictive model that compares new instances with the existing classes (normal or abnormal traffic) and decides upon each event accordingly.

Supervised machine learning is defined by the use of labelled data for training [5][6], that will help classify or predict accurately when the model is used. By the term label, it is implied that the training dataset includes input data and the corresponding outcome each entry should generate. As input data is fed to it during training, the model adjusts its weights (interconnections inside its nodes) so that the outcome it produces matches the correct outcome as much as possible. This is measured through the use of a loss function that calculates the deviation between the produced result and the correct result. The goal of training is to minimize the loss function.

There are two categories that supervised learning applies to: classification and regression. In regression problems, the model needs to understand the relationship between the dependent values and the independent ones. It is usually applied when we want to make future projections, like weather prediction, stock prices, business revenue, etc. On the other hand, in classification applications, the model needs to understand what features make an instance the class it is, and assign the input data into the right categories, like the spam folder of our emails.

Anomaly detection is a classification problem, and some of the most common supervised learning methods for classification are the ones that were used in our project, which are analysed in *section 3. Classification models analysis.*

Supervised learning differs fundamentally from unsupervised learning, because, unlike with unsupervised learning, it uses labelled data. Unsupervised learning methods try to discover patterns in the data and cluster them or make associations. Each method has its own advantages and disadvantages, but let's take a look at what these are for supervised learning methods.

Disadvantages and challenges of supervised learning:

- Creating labels for some datasets can be time consuming, or even impossible in some cases, due to the limited information available on the data.
- Irrelevant input features can hinder the performance of the model greatly, as well as when unlikely, incomplete, or out of bounds values are inputted as training data.
- When dealing with classification applications, representing all classes in a balanced way is a challenge and the performance of the model is lowered if the data is imbalanced.

That is especially true for big data analytics, where classification is sometimes impossible.
- Models can be prone to overfitting, when the quality of the training data is not good enough, or when the hyperparameters of the model are not optimal.

Advantages of supervised learning:

- When prior knowledge and experience is important, supervised learning is the best way to create a model based on those characteristics, which will learn from experience.
- Supervised learning helps optimise the performance of our model based on what features of the input data are selected.
- It is helpful in various real-world computational problems that other methods are incompetent in, due to the lack of information during their training phases.

Supervised learning is the most reliable way to create models, when it is important to know how well the algorithm used performs, how accurately it works. They are the best predictive models for many applications, but on the other hand, they require a lot of preparation and pre-processing of the training data so that their results are not biased or overtrained.

For anomaly detection, the problem with supervised training is that the process of assigning labels to traffic data is very time consuming and even impossible when dealing with unknown or novel attacks. However, unsupervised methods are not dependable as to their performance, because there is no way for the model to validate whether the traffic is really normal or not.

2.2. Anomaly detection with machine learning: related research

There have been many innovations in the field of anomaly detection in the past few years, using the NSL-KDD dataset. There are many unsupervised learning experiments, with autoencoder and one-class SVM combinations [2]. Convolutional autoencoders are used by [7] paired with a one-class SVM layer that classifies the data after the convolutional step of the model. In [2] we can also find that one-class SVM as well as autoencoders have been also used in self-supervised learning methodologies. One-class SVM has also been paired with Bidirectional LSTM methods in [8]. Neural networks have been used extensively in anomaly-based intrusion detection, as is evident in [3], and DNNs have been tested with selective feature extraction [9].

There have been comparative studies such as our own too, in the past couple of years, namely [10][11][12][13], and we can also find more comparisons of such research projects in review studies [3][2]. Going beyond the NSL-KDD dataset there are many more articles, but for a consistent view of this research topic, we will stay with those projects that use the NSL-KDD as their dataset here. In section *5.2. Evaluation and results compared to relevant research* there is a more thorough comparison between the methods that were developed in this thesis, and the state-of-the-art studies conducted lately, after our own results are extracted.

2.3. The advantages of the NSL-KDD dataset

Lastly for this section, it is worthwhile to mention some more information on why the NSL-KDD was chosen and where it came from. The NSL-KDD was created in 2009, as an effort to overcome some of the limitations and problems that its ancestors, DARPA (1998) and KDDCup99 (1999), had. it is, like the original KDDCup99 before it, a publicly available dataset of network traffic data records, which contains a selected subset of the data in KDDCup99 [1]. The selection of that data occurred by applying some filters targeting the problematic instances in it, and at the same time, providing best practices for data mining to create the new dataset. So, the main advantages of using this dataset are:

- It doesn't include any redundant records in it, thus avoiding biasing toward more frequent records.
- There are no duplicate records in the test set, so that the performance of the models is not biased by those with falsely higher detection rate.
- The number of selected records from each difficulty level is inversely proportional to the percentage of records in the original KDDCup99, therefore the classification rates of various machine learning methods vary in a wider range.
- Opposite to the KDDCup99, that had millions of data records in it, both the *KDDTrain+* and the *KDDTest+* have a reasonable amount of records in them, making it affordable to run experiments on the complete datasets instead of selecting a random small portion of it. That is why evaluation results of different research groups are consistent and comparable (like it happens with our models).

The NSL-KDD is not a perfect dataset, as it is quite outdated, and because it is a synthetic dataset. There is, however, much value in those rare, good datasets that are available, even if they are old. Firstly, they are already labelled, a process that is very time consuming or even impossible sometimes, which allows researchers to test supervised learning methods, or validate the unsupervised models more frequently used today. Benchmark datasets, like NSL-KDD, are used for validation and evaluation of new approaches to intrusion detection, and comparison between different methods, old and new. They are also the only way to have repeatability in the experiments done over the years, especially because they are publicly available to all researchers. A rich in features dataset like NSL-KDD also allows different approaches to fine-tune into different parameters, and extract features for more light-weight models, or simply provide a base on which new datasets can be built.

The network traffic datasets are valuable assets for IDS research. However, none of them can clearly represent the real-world traffic, as it is constantly evolving, and new attacks always appear (or haven't been discovered yet). Apart from the privacy and security concerns that hinder the mining of real data, simulations are also difficult to do realistically. Evaluation of IDS datasets is challenged by all the difficulties in collecting attack and victim scripts, by the rapid speed at which attacks evolve and are produced, and also by the many different network services that not only make traffic more complex, but also leave new gaps for exploitation.

3. Classification models analysis

In this section, the algorithms that were used for the classifications are going to be described and briefly analysed, to better understand the way they work and what the advantages and disadvantages of their use are.

3.1. Logistic Regression

Despite its name, logistic regression is a supervised classification algorithm, one that uses regression to calculate the probability that a specific data entry (input – X_i), belongs to category Y_j. Describing it first as a binary classification problem, for an easier approach, will help us understand the mechanics of this algorithm, while its use can easily be expanded for multiclass classification problems, as multiclass classification (multinomial logistic regression) takes place the same way as binary, in a one-against-all way; this means that the class examined is classified as 1 whereas all other classes are considered 0 for the test ($g(z)$) of each specific entry.

The function that logistic regression uses for the calculation of the probability is the sigmoid function [14]:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Equation 1: logistic/sigmoid function

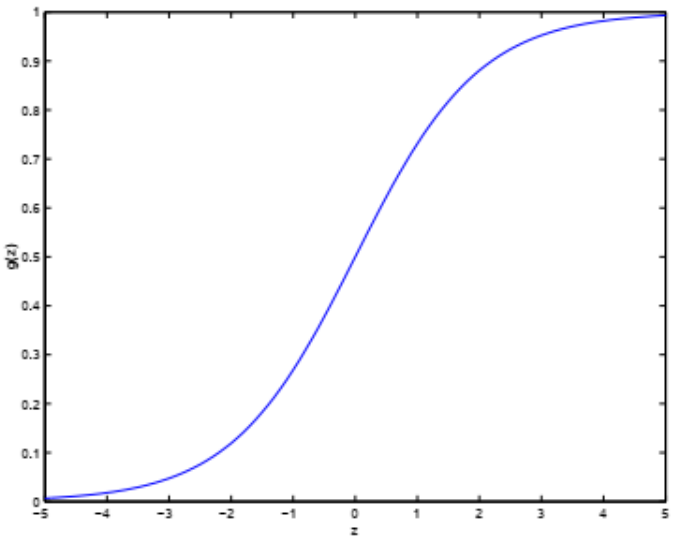


Figure 1: sigmoid function graph (source [14]*)*

It is noticeable from the graph of the function that when $z \to \infty$, then $g(z)$ tends toward 1 and when $z \to -\infty$ then $g(z)$ tends toward 0, which is why regression works well as a function.

The variable z describes the input value, which is the variables vector of the entries $X_i = \{x_0, x_1, \dots, x_{n-1}\}$ (for n number of features in the dataset) multiplied by weight values, that will be tweaked as the model tries to predict y with respect to X_i.

$$y(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_{n-1} x_{n-1} = \sum_{i=0}^{n-1} \theta_i x_i = \theta^T X$$

Equation 2: output y as a function of the input values X

Thus, in the case of logistic regression, this abstract function becomes $y = g(\theta^T X_i)$. Through the training of the model, the weight values (θ^t) are randomly initialized and then change so that the loss function is minimized, and this sets the threshold for which $y = 1$ or $y = 0$.

Logistic regression is one of the simplest machine learning algorithms, so it doesn't need many conditions to generate satisfactory results and doesn't require much CPU power usually. It also doesn't overfit as much as more complex algorithms and can easily update with new data. Nevertheless, its simplicity hinders its performance on higher dimension datasets, and highly correlated variables in a dataset should be avoided; also, it needs to train with larger datasets without redundant records in them [15].

3.2. Decision Tree

The decision tree classifier is a tree-shaped algorithm that is commonly used for classification applications.

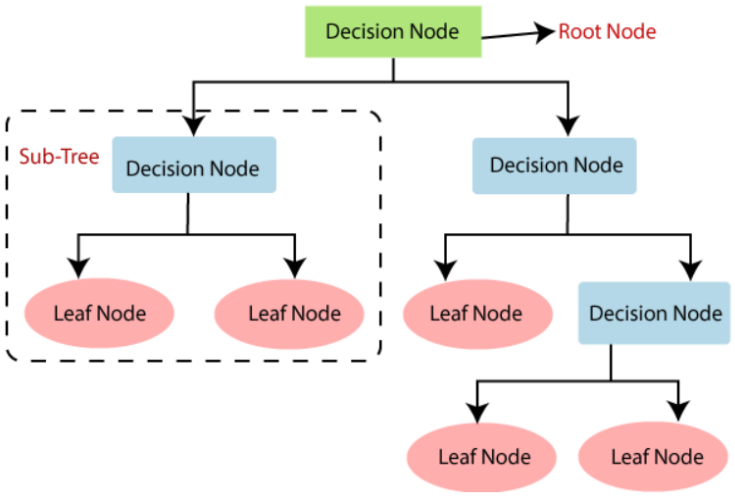


Figure 2: decision tree classifier representation (source [16]*)*

The *root node* represents the beginning of the decision tree and includes the whole dataset. It gets further divided (*splitting*) as the algorithm poses conditions to the dataset that create sub-classes according to the outcome of each entry. Through the splitting process, *branches* are created, as different classes of data follow different paths. The *leaf nodes* represent the

outcomes of the classification process, when the model cannot further classify the subset that has gone that way. Another process for correcting the model and minimizing error is *pruning,* which cuts out the branches that don't have any data and keeps the optimal tree paths [16].

The decision tree is a simple algorithm that mimics the way humans make decisions, so it can be very useful in decision-related problems, and its simplicity also requires less cleaning and preparing for the data. However, when the dataset contains many labels, the classifier is prone to overfitting, and its complexity becomes very high when there are many layers to the decisions.

3.3. K – Nearest Neighbours

K nearest neighbours is one of the most essential supervised classification algorithms in machine learning. It is also one of the most basic ones, given that it doesn't make any assumptions about the distribution of the data (non-parametric algorithm). It finds application in pattern recognition, intrusion detection and data mining [17], [18].

As a supervised method, the training set is first distributed according to the labels in a n dimensional space (as the vector of the input features enforces), like we can see the two labels ("Green"/"Red") in *Figure 3*:

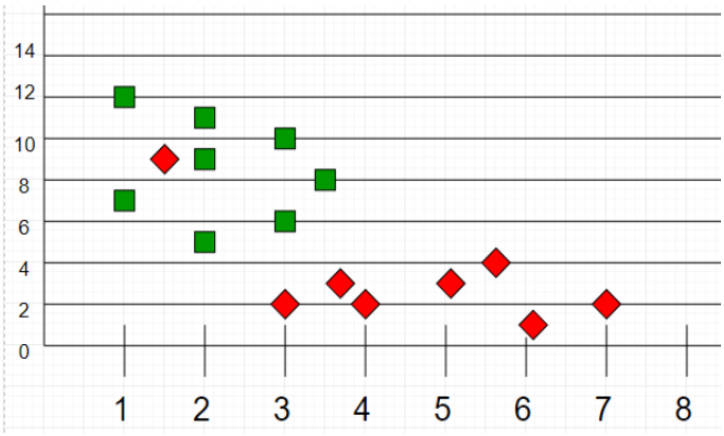


Figure 3: knn distribution of the training set according to its labels (source [17]*)*

After that, during the testing, unclassified data is placed in the graph according to their attributes, and the model must try to classify it properly *(Figure 4)*. This is where the parameter k plays an important role, as this algorithm determines the class of each test datapoint as the same class that the majority of its k- nearest neighbours are, through a voting mechanism. If we set $k = 1$, then the unclassified datapoint will be grouped together with its closest classified point. In general, when we choose fewer neighbours, it is better to choose an odd number of them, so that there is no conflict to resolve.

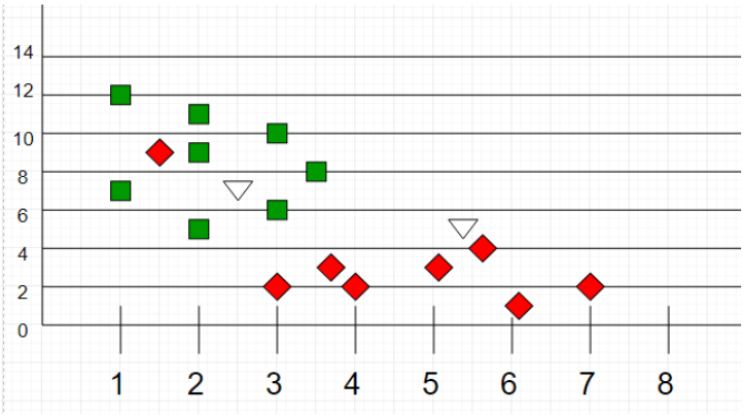


Figure 4: knn during testing

K- nearest neighbours is a good algorithm to use when the data is multinomial (multiple classes), or non-linear (for regression problems), since it doesn't have underlying assumptions on the training data distribution; it is also easy to understand and implement. However, the computational cost and memory requirements are relatively high, as it must store all the training data to work, and if the value of k is high, then the voting process takes much longer to predict the outcome.

3.4. Gaussian Naïve Bayes

Naïve Bayes is a probabilistic classification algorithm based on the Bayes theorem. Gaussian NB is an extension of it, which assumes Gaussian (normal) distribution of the data. The naivety of the model comes from the assumption that all the features of the dataset are independent from each other, meaning that variation in one variable of the dataset do not impact the other features. The Bayes theorem is a conditional probability theorem that defines a classifier so that the error rate (misclassification) is minimized through the training phase, that works in a way to go from $P(X|Y)$ to find $P(Y|X)$ [19][20].

In the Bayes rule, from the training data we have:

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$$

Equation 3: probability that X belongs to class Y

And from this, which is learnt during training, the model needs to learn the opposite (if Y is the correct class), during the testing phase:

$$P(Y|X) = \frac{P(X \cap Y)}{P(X)}$$

Equation 4: probability that class Y is the correct outcome of occurrence X

There can be many mathematical expressions for the probability, but one of the most reliable ones, which is commonly used with Naïve Bayes, is the Gaussian distribution (aka normal distribution), which assumes that X is a continuous variable:

$$P(X|Y=c) = \frac{1}{\sqrt{\pi\sigma_c^2}} e^{\frac{-(x-\mu_c)^2}{2\sigma_c^2}}$$

Equation 5: *probability of X being class Y when X follows Gaussian distribution*

In this *Equation 5,* σ is the variance, μ is the mean value of the data, and X is calculated for a given class c of Y.

The Gaussian Naïve Bayes is a simple, fast, and very effective algorithm, that can even outperform high complexity models. It can predict multiclass datasets, especially of categorical labels, and can perform well with less training data than other algorithms, as long as the condition for independence of the variables holds. On the other hand, the probabilistic nature of the algorithm comes with many conditions. If the input variables are not independent (which is rarely the case in real life) then the model underperforms significantly, as we will see in our own results too. Another big problem is that if a class that is present in the test set has not appeared in the training set, then the model assigns that class zero possibility.

3.5. Multi-Layer Perceptron

The multilayer perceptron (MLP) is a fully connected Artificial Neural Network (ANN). it feeds the input form the *input layer* to the *hidden layer* by taking the dot product of the input values with the weight parameters that exist at the interconnection of all nodes of the ANN. When all the input nodes are weighted, they add up at the entrance of the next layer's node, where their resultant value passes an activation function (e.g., sigmoid, ReLU, tanh).

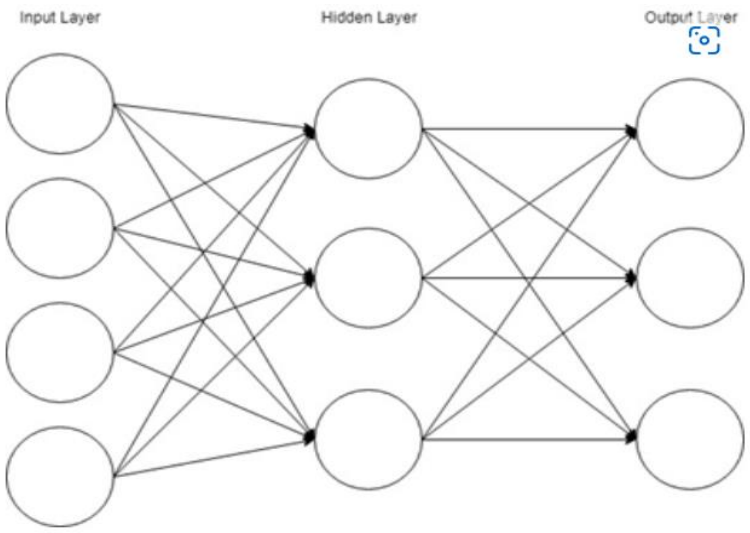


Figure 5: *the multilayer perceptron, a fully connected feedforward ANN*

The value that comes out of the activation function is now the value of the hidden layer's node and it can be fed to the next layer (another hidden layer or the output layer) with the same process of dot product calculation and activation function; this process is repeated in all the nodes of all the hidden layers [21], [22].

After getting to the output layer, the outcome value of the ANN is either used for backpropagation during the training phase, or the is presented as the result of the prediction during the test.

Multilayer perceptrons are the basis of all ANNs, and have greatly improved machine learning algorithms, both in regression and in classification applications. Their flexibility and the abundance of both activation and optimisation functions have enabled computers to not be constricted by XOR calculations and enrich their learning potential for more rich and complex problems.

MLPs can be shallow, when there is only one hidden layer, or Deep Neural Networks (DNNs), when there are two or more hidden layers. ANNs, especially DNNs are at the forefront of research in the past few years, as they are fundamental for deep learning and AI. One of their core strengths is that they solve problems stochastically, therefore allow approximate solutions to very complex or even unsolvable problems. The stochastic way of work allows the model not to make any assumptions about underlying probabilistic density or other relations between the variables of the input data, but rather get to it through the weight functions (interconnections of the nodes) and the repetitive process of training. MLPs can have high performance scores even with less training data, if given a sufficient number of nodes and layers, and a two-layer backpropagation neural network with enough hidden neurons has been proved to be a universal approximator [23]. The most important disadvantage of MLP compared with other DNN methods is that it is fully connected and creates a dense network, so the number of parameters needed for the model becomes very high. This leads to inefficiency and redundancy in more complex problems [24].

These are the five models that were used for this project. This brief explanation of the way they work is hopefully helpful for understanding why each of them showcased the results it did during the experimental phase. Each model has its own use and advantages in different cases, so it was considered useful to provide this comparison among them, with the NSL-KDD, which provides for a rich dataset when it comes to the number and variety of features, variation in the correlation between them, and many classifications to study.

4. Characteristics and pre-processing of the NSL-KDD dataset

The NSL-KDD dataset is devised of many subsets of data. Specifically, the main sub-datasets are the *KDDTrain+* and the *KDDTest+*, which have $125{,}973$ and $22{,}544$ rows respectively, giving it a 17.9% rate of test to training data. These two contain the full training and test datasets in .csv format, including attack type labels for each record and a difficulty level (ranging from 1 to 21). Apart from the two, the set contains a subset of the test including only the records with difficulty level lower than $21/21$ named *KDDTest-21*, and another subset of $25{,}192$ records, randomly taken from the training dataset, the *KDDTrain+_20Percent*. The records of *KDDTest-21* and *KDDTrain+_20Percent* are all included in the bigger datasets, *KDDTest+* and *KDDTrain+* respectively, so all the information of the dataset is present in the main files.

The datasets are comprised of records of network traffic, as seen by a simple IDS network. Each record (row) has 43 features (columns), out of which the first 41 $(\#0 - \#40)$ are characteristics of the traffic, $\#41$ is the attack label, and $\#42$ is the difficulty level of the input.

4.1. The attack labels (traffic type)

In total, there are 39 attacks (40 different labels including the normal traffic) that belong in four classes: Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R) and Probe attacks. The four classes of the NSL-KDD dataset are different in their objectives, the way they infect the network, and how they are distributed in the dataset. Also, a fifth category of the dataset is the normal traffic, which, naturally, is encountered more than all the attack traffic.

Denial of Service (DoS): DoS attacks flood the network with abnormal traffic, so that the normal traffic can't reach it. As a result, the network will most likely shut down, in order to be protected from the volume of data trying to pass through the IDS.

Remote to Local (R2L): as the name suggests, R2L is an attack that tries to get local access to a system or network from a remote machine that can't normally do that, so the attacker tries to "hack" their way into the network.

User to Root (U2R): this is an attack where a normal user account tries to gain privileged access as a super-user (root access), by exploiting vulnerabilities and gaps in the devices of the system or network.

Probe: probe or surveillance attacks try to steal information from a network. That can be client information, banking data, passwords or other personal data that are passing through the network.

In the NSL-KDD, these four classes (as well as normal traffic) are not equally distributed in the dataset. We can see from the initial analysis that the most common class of attacks is the DoS, both in the training *(Figure 66)* and test set *(Figure 87)*:

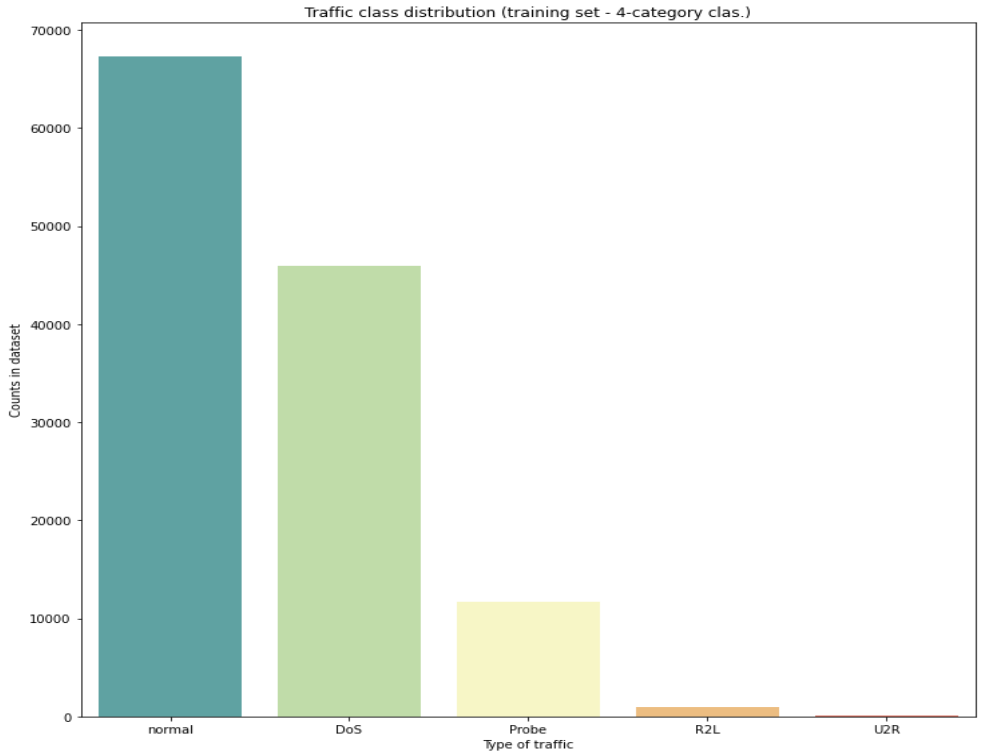


Figure 6: distribution of traffic by class of attack in training dataset

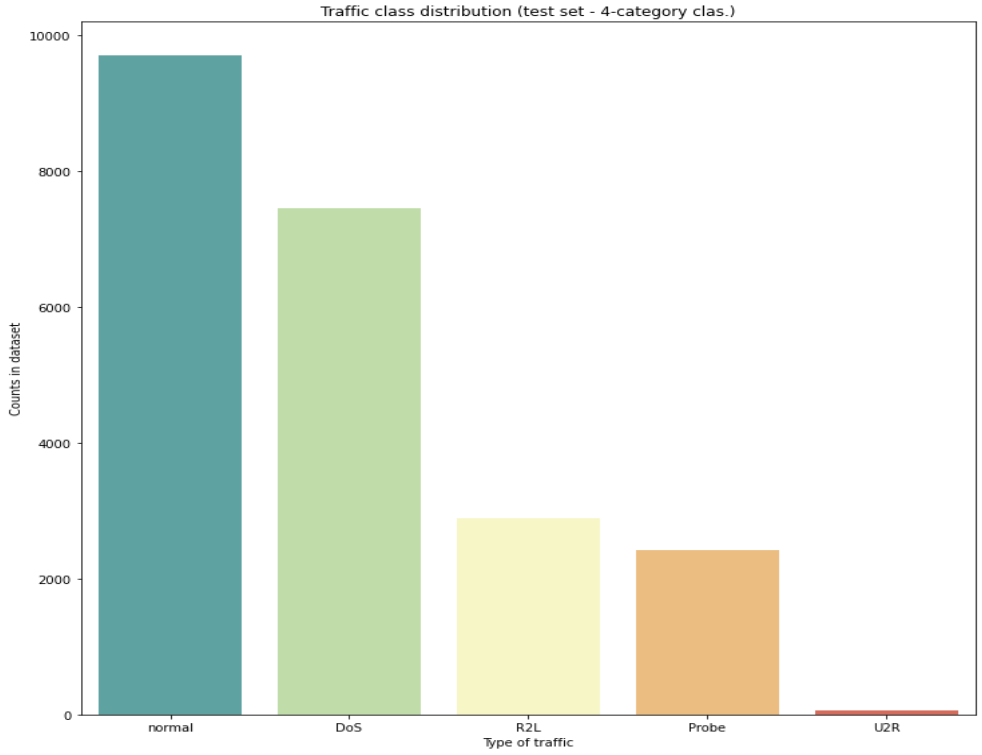


Figure 7: distribution of traffic by class of attack in test dataset

Seeing it in absolute numbers *(Table 1)* there are a few differences between the training and the test set.

Table 1: distribution of traffic in training and test datasets

Type of traffic	# in training set	% in training set	# in test set	% in test set
normal	67343	53.46%	9711	43.08%
DoS	45927	36.46%	7460	33.09%
Probe	11656	9.25%	2885	12.79%
R2L	995	0.79%	2421	10.74%
U2R	52	0.04%	67	0.30%

In total, we see a skewed distribution towards the normal and DoS traffic in both datasets. In the test set however, the normal traffic is not even half of the total and the R2L class of attacks is accordingly boosted, compared with the training set. This uneven distribution of internet traffic is a realistic representation of typical internet traffic, where DoS attacks are the most common, followed by probe attacks, while R2L and U2R are hardly encountered in real life.

For a more detailed approach, all the different attacks need to be addressed. It is notable that DoS attacks are the most common in terms of encounters in both the datasets, but when it comes to the number of different attacks each class includes, R2L is the one that comes first. In the following *Table 2*, we can see all the labels that are included in the NSL-KDD, divided in their classes:

Table 2: all attack labels of the NSL-KDD, by class

Class	R2L	DoS	U2R	Probe
Attacks	ftp_write guess_passwd httptunnel imap multihop named phf sendmail snmpgetattack spy snmpguess warezmaster warezclient xlock xsnoop	apache2 back land neptune mailbomb pod processtable smurf teardrop udpstorm worm	buffer_overflow loadmodule perl ps rootkit sqlattack xterm	ipsweep mscan nmap portsweep saint satan
Total	**15**	**11**	**7**	**6**

The dataset was also studied to classify all the attacks separately, so it was worthwhile to investigate how many encounters of each attack are found. This can be seen in the next couple of figures *(Figure 8, Figure 9),* in the training and test set respectively. In the case of studying the various attacks separately, the normal traffic outnumbers the rest by far, followed by Neptune, the most popular DoS attack. It is also notable that in the training set, only 23 labels

(22 attacks and 1 for normal traffic) are encountered, whereas in the test set, there are 38 different labels, to make sure that the IDS can identify attacks that were not previously seen during training, when it first encounters it during the validation of the model.

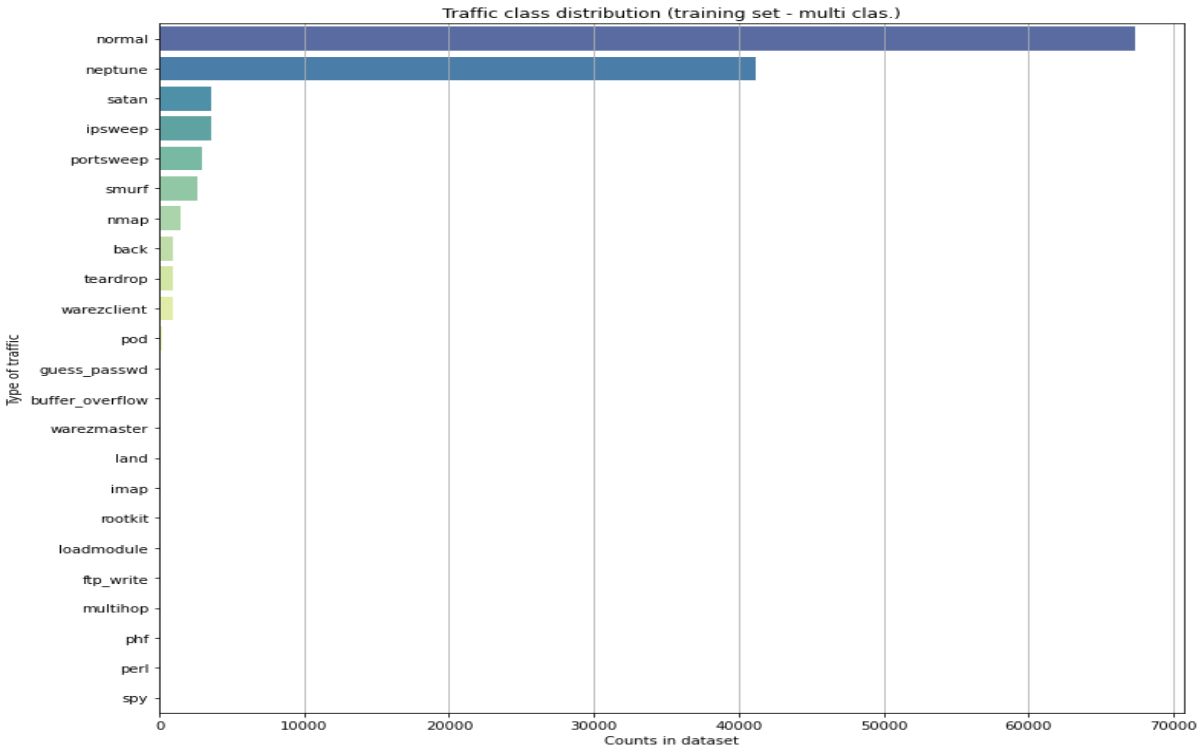


Figure 8: traffic distribution in training set

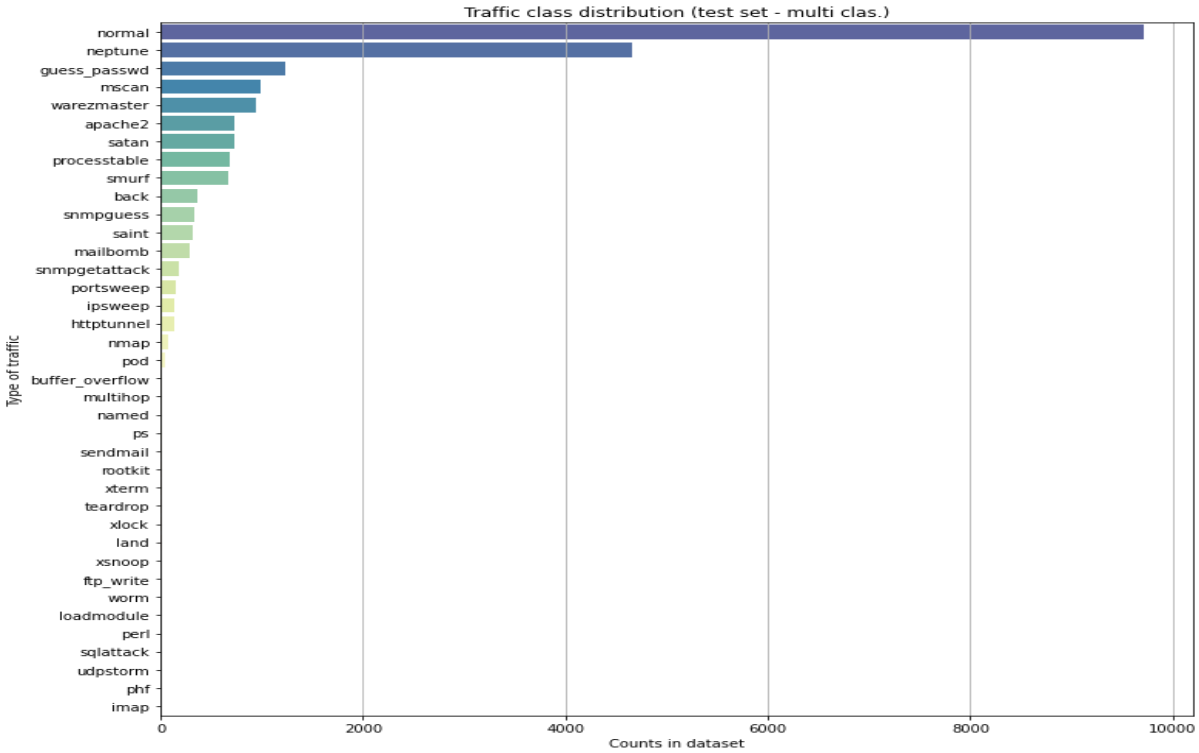


Figure 9: traffic distribution in test set

Figure 10*10* and *Figure* 11*11* showcase the rate between normal and "abnormal" traffic, since the dataset was also studied as a binary classification problem. It is noticeable that in the test set, there is more abnormal traffic than there is normal.

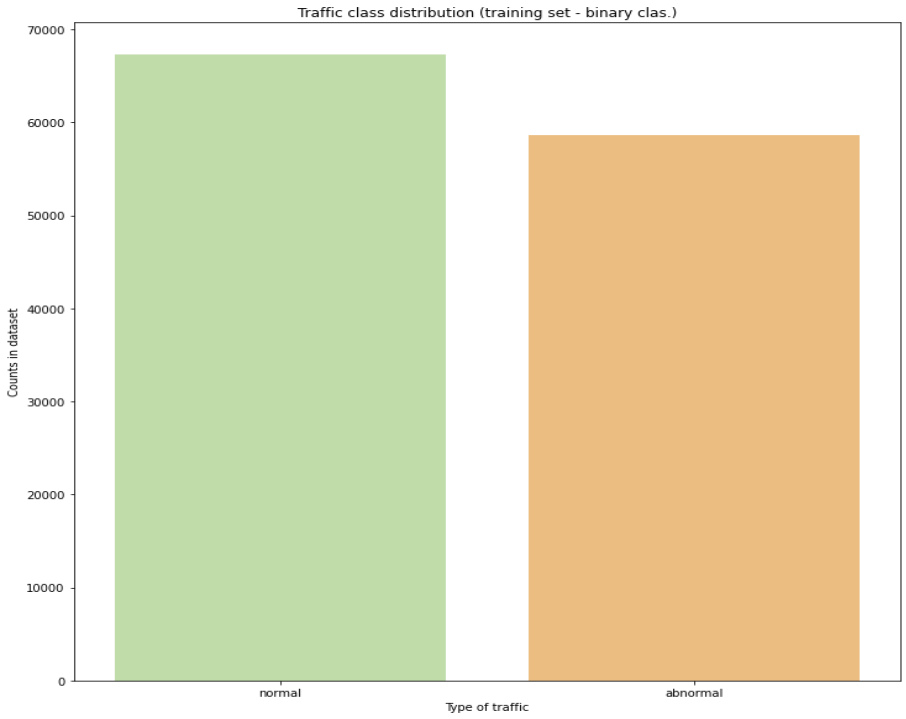


Figure 10: traffic ratio in training set

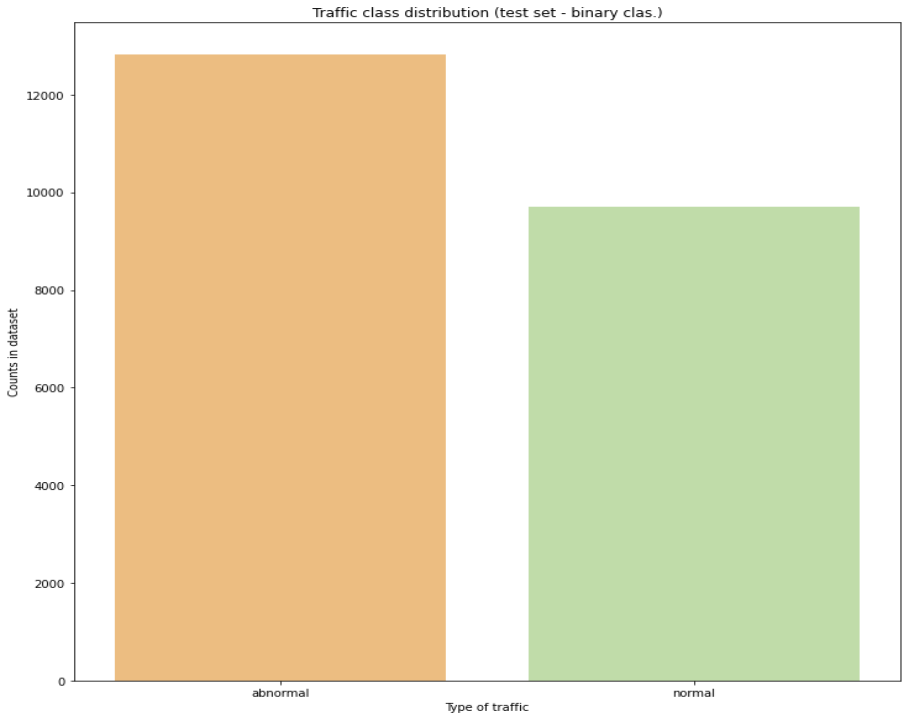


Figure 11: traffic ratio in test set

4.2. The features of NSL-KDD

Other than the label feature of the dataset, which is found in the $42nd$ column $(\#41)$, and the last column that is the severity/difficulty level ($\#42$, which is removed during the pre-processing phase), the rest of the features represent the information that the IDS uses to determine the type of traffic and assign the appropriate label to it. These features can be categorised by the information they contain, and the way it is extracted from the packets arriving at the network.

There are four categories by which the features are grouped [25]:

Intrinsic features (columns $\#0 - \#8$), that contain information from the header without needing to dive into the payload, which hold the basic information about the incoming packet.

Content features (columns $\#9 - \#21$), that hold information about the incoming packets in a connection-based way that allow the IDS access to the payload.

Time-based features (columns $\#22 - \#30$) have the traffic analysed over a 2 second window, and mostly contain rates and counts (e.g., of connection attempts, port number, connections that activate certain flags, etc.) rather than information from the packets themselves.

Lastly, *host-based* features (columns $\#31 - \#40$) are similar to the last category, but instead of analysing inside the 2-second window, they gather information over a series of connections made (e.g., percentage of connections with the same destination host address/port number), in order to access attacks that span longer than the window allowed previously.

A table of all the 41 features, with a brief explanation of each, can be found in *Annex A: table of the NSL-KDD features.*

The NSL-KDD dataset has different kinds of data in its features, which makes it necessary to pre-process the data, to be able to find correlation and investigate it, or feed the data into a model. More specifically, there are four types of data in the dataset: *categorical* (columns $\#1, \#2, \#3, \#41$), *binary* (columns $\#6, \#11, \#13, \#19 - \#21$), *discrete* (columns $\#7, \#8, \#14, \#22 - \#40, \#42$) and *continuous* (columns $\#0, \#4, \#5, \#9, \#10, \#12, \#15 - \#18$). Binary, discrete, and continuous values, being numerical, are okay to be left as they are, but the categorical values, being in strings forms, are not suitable for further analysis and finding relationships among the data.

4.2.1. Categorical features

The categorical variables found in the dataset, apart from the attack label column $(\#41)$ that has already been investigated, have to do with three features of the connection: protocol (col. $\#1$), service (col. $\#2$) and flags (col. $\#3$). In this section, each of these features in the dataset is going to be briefly analysed, to get an idea of what the network traffic looks like, what is more

common and how the following correlations (during the pre-processing phase) can be explained.

A list of all the services can be found in the NSL-KDD can be found in *Annex B: table of all the services in the NSL-KDD dataset.* In this section, a brief presentation of the protocols and services is given, to present a picture of the characteristics of the dataset.

The protocols recorded in the dataset all belong to the transport layer of the OSI model and of the TCP/IP stack [26], and the network layer (in OSI) or internet layer (in TCP/IP). The transport layer, which is the most represented in the dataset, is responsible for *process-to-process delivery* (by port number addressing), *end-to-end connection* between hosts, connecting devices without considering the network fabric, *multiplexing and demultiplexing*, so that different applications are simultaneously used over the network, *congestion* and *flow control*, and *data integrity/error correction.*

In the NSL-KDD, three protocols are found:

TCP (Transmission Control Protocol): TCP is the most popular protocol of the transport layer because it provides reliable transmission of all packages. It does so, by having an acknowledgment signal for all received packets, and it resends the lost ones. While this is a great advantage that provides a reliable and safe communication, it adds an additional overhead due to these features. It is commonly used by protocols such as HTTP and FTP.

UDP (User Datagram Protocol): UDP, unlike TCP, doesn't provide acknowledgement of the received packets, thus the connection is not reliable, it relies on a "best effort" approach. However, it is very simple and comes with much less overhead compared to other protocols. it is most commonly used in streaming/real time services, such as video or voice streaming.

ICMP (Internet Control Message Protocol): ICMP is a network/internet layer protocol, despite sometimes being perceived as a transfer layer one, as the internet layer depends on ICMP for error and control messages (ping, traceroute, destination unreachable, etc.). It is mainly used to determine whether or not data has reached its intended destination in a timely manner. In the case of the NSL-KDD dataset, and TCP data dumps in general, ICMP is usually seen when the packets are fragmented.

In the NSL-KDD dataset we can find most of the traffic using the TCP protocol, a smaller percentage using UDP, and a small number of records being ICMP messages, with both the training and the test sets behaving similarly *(Table 3, Figure* 12*12 and Figure* 13*13)*:

Table 3: protocols in the NSL-KDD subsets

Protocol	# in training set	% in training set	# in test set	% in test set
TCP	102689	81,52%	18880	83,75%
UDP	14993	11,90%	2621	11,62%
ICMP	8291	6,58%	1043	4,63%

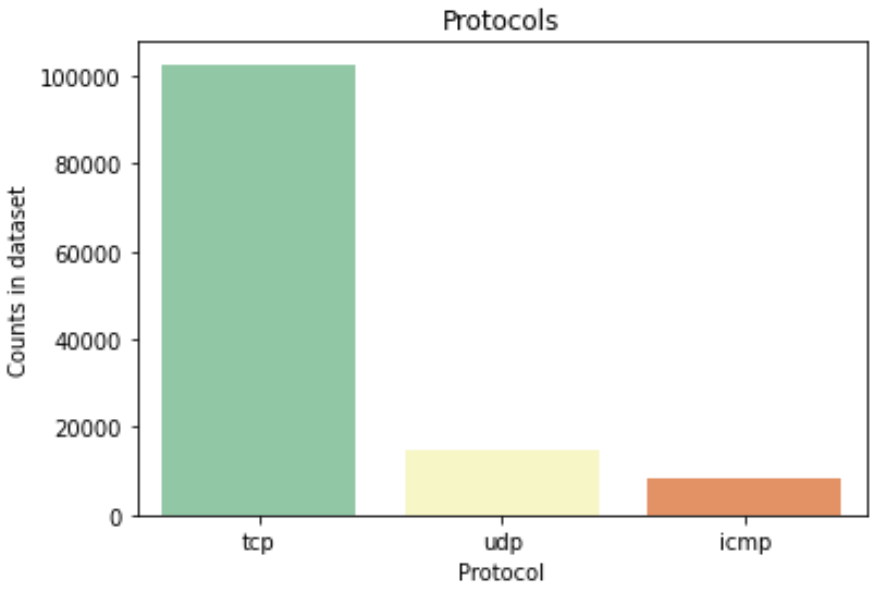


Figure 12: protocols distribution in training set

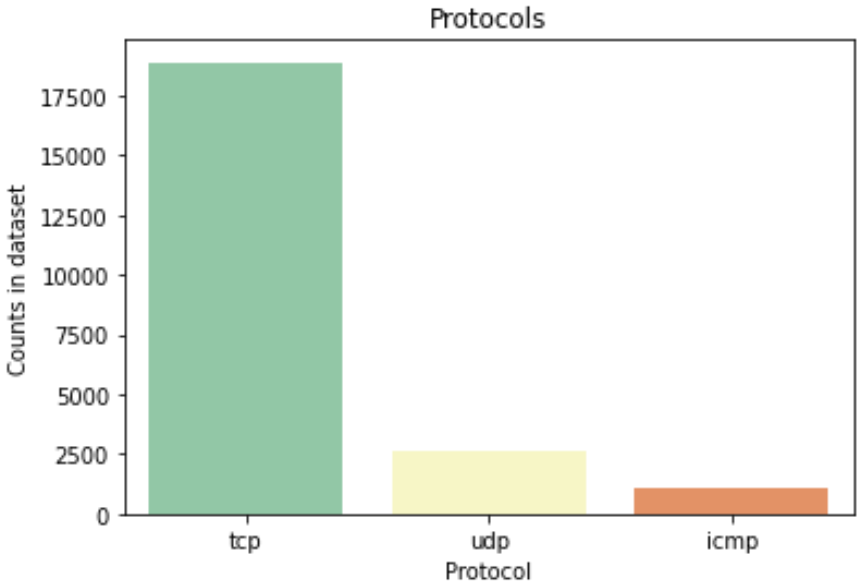


Figure 13: protocols distribution in test set

The services (column #2) are all about the application layer (top level in both the OSI and TCP/IP models). In the dataset, there are protocols that enable capabilities such as email exchange, website navigation, data storage and manipulation, DNS, etc. and work in a *server-client* or a *peer-to-peer* philosophy.

Since the NSL-KDD is labelled with 70 different services, a list of them and their encounters in the datasets is going to be given in Annex B. Below, are the diagrams produced from counting all the featured services, to get an idea of what applications are the most common in the network. We can see that, in both the training *(Figure* 14*)* and test set *(Figure* 15*)*, http (communication between web clients and servers) and private network (e.g. VPN) traffic accounts for about half of the total traffic, followed by domain requests and telnet respectively.

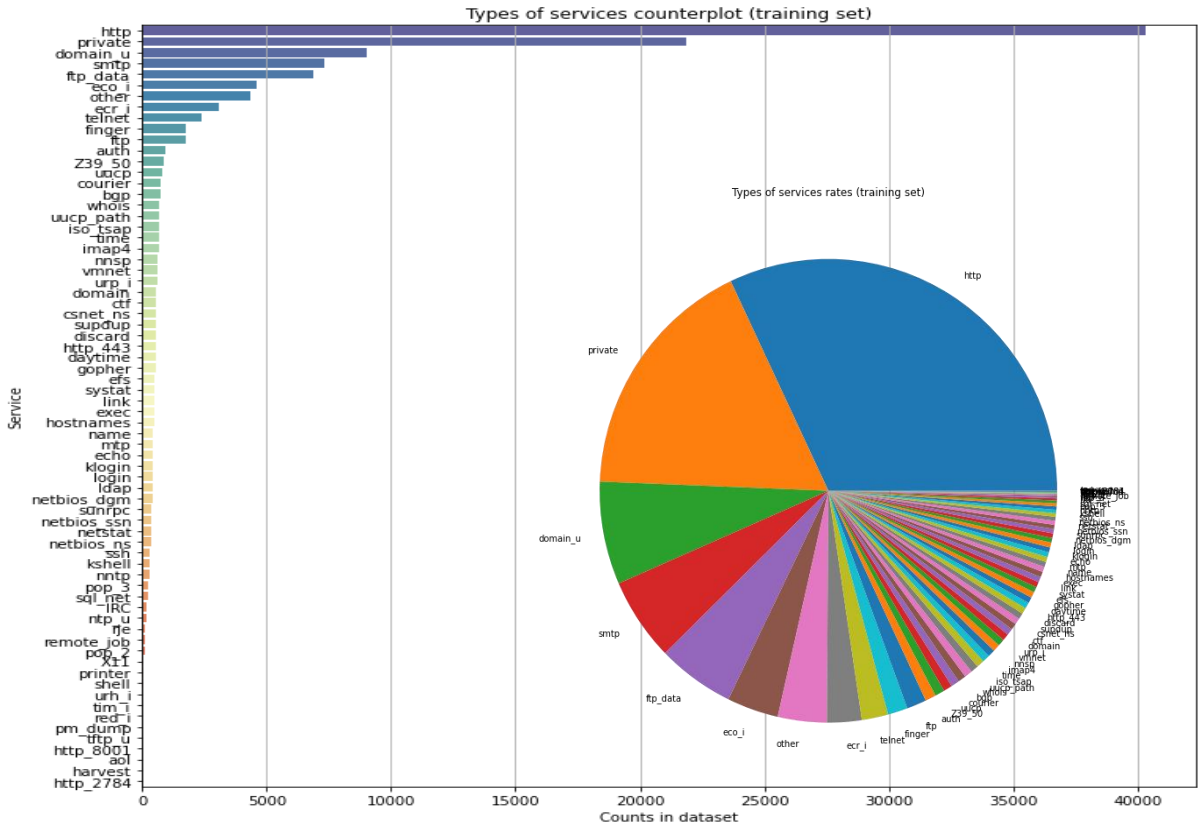


Figure 14: services distribution in training set

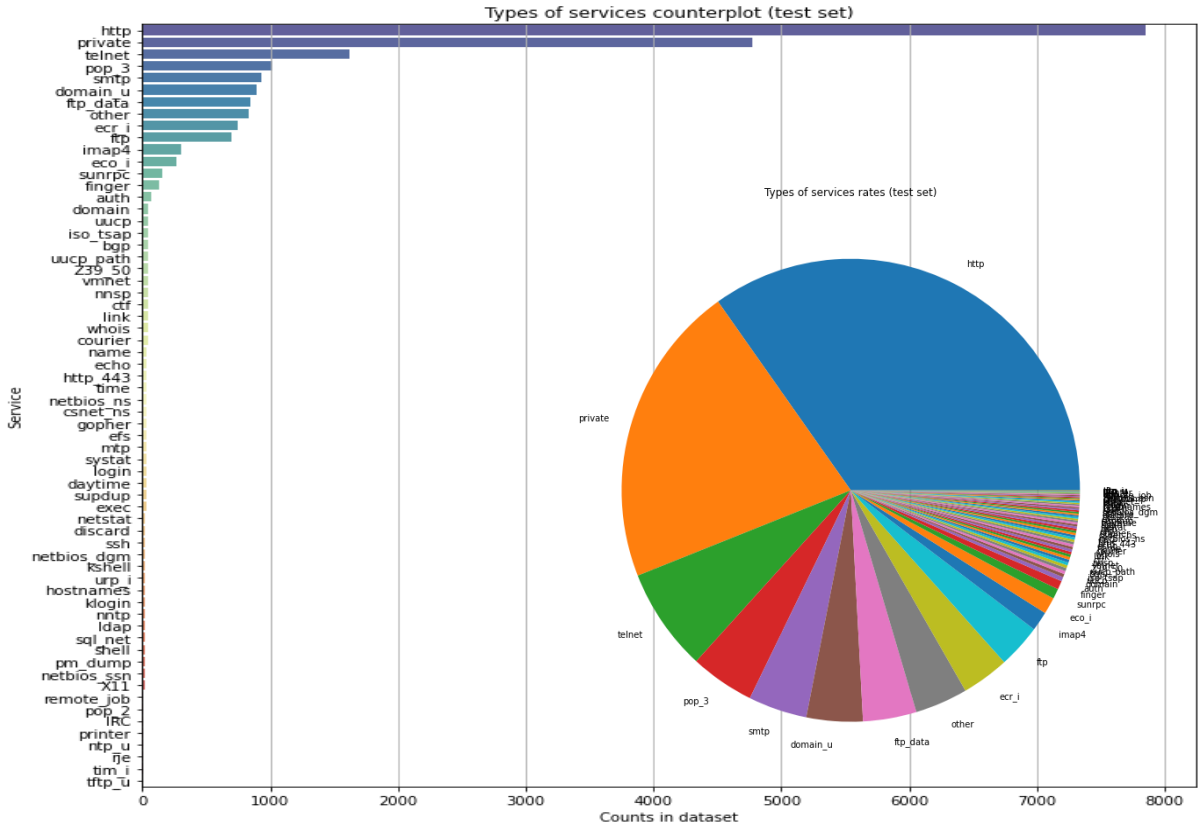


Figure 15: services distribution in test set

Flags are an important feature of the connection, as they describe whether the connection was established, terminated, or rejected normally. In the NSL-KDD dataset, there are 11 different flags that are found in both the training and the test sets. It is shown in the figures below *(Figure 166, Figure 177)* that the two subsets have different distributions of the flags:

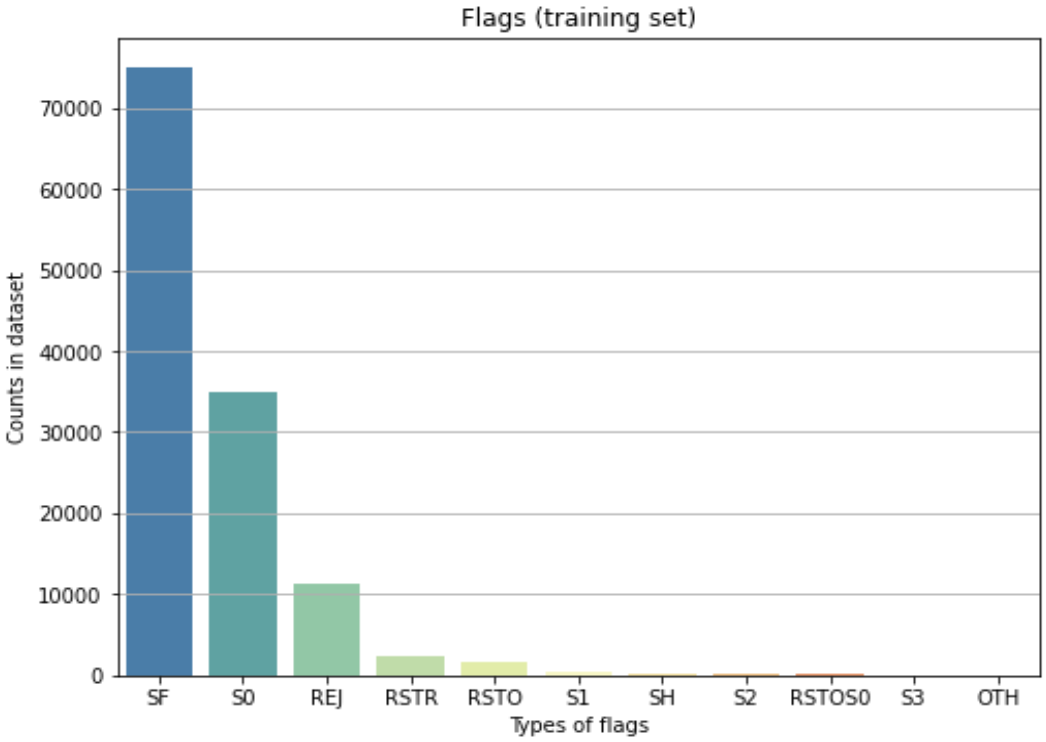


Figure 16: flags distribution in training set

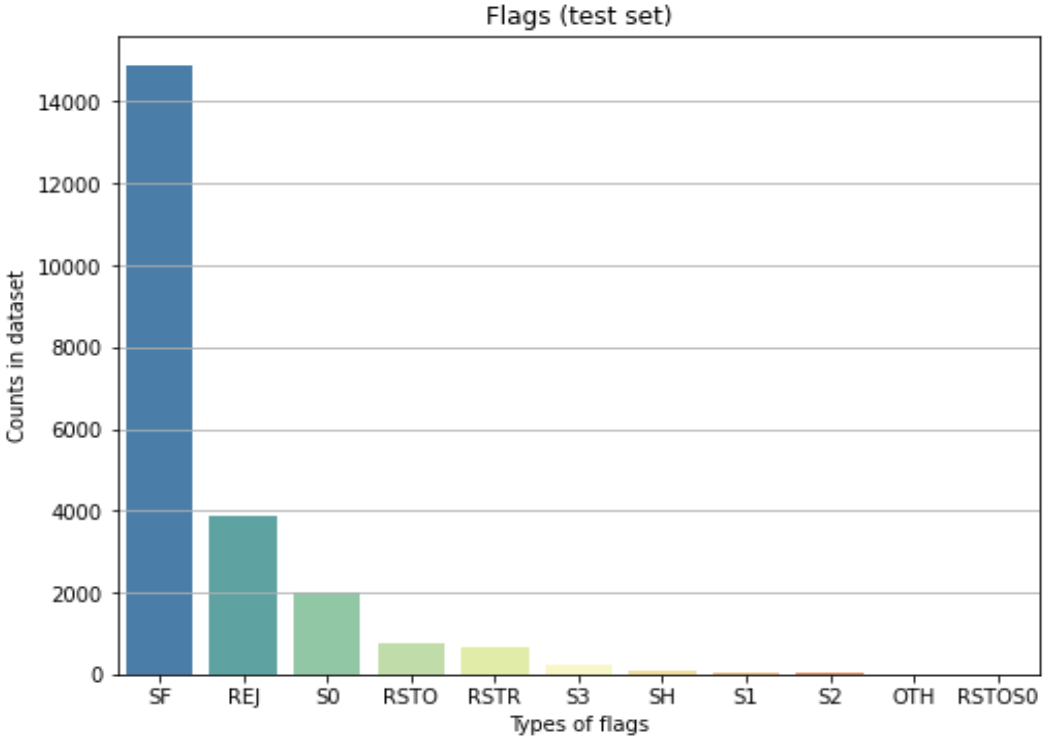


Figure 17: flags distribution in test set

The difference between the ratio of REJ and S0 flags, which are the most prominent after the SF flag, can be understood after looking at what each label means, in *Table 4*:

Table 4: flags in the NSL-KDD dataset

Name	Meaning
SF	Normal establishment and termination
S0	Connection attempt, no reply
REJ	Connection attempt rejected
RSTR	Connection reset by the destination
RSTO	Connection reset by the source
S1	Connection establishment, no termination
SH	Source sent a SYN and FIN, without a SYN-ACK from the destination
S2	Connection established, close attempt from source but no reply
RSTOS0	Source sent a SYN and RST, without a SYN-ACK from the destination
S3	Connection established, close attempt by destination but no reply
OTH	No SYN, just midstream traffic that is not later closed

In the test set, where the abnormal traffic is higher, it is natural to have more rejection flags, whereas in the training set, where the normal traffic prevails, more connection attempts are to be expected.

4.3. Pre-processing of the NSL-KDD dataset

One of the most important steps in creating a data science model is pre-processing the data. Python is a language especially capable of handling tasks that have to do with data handling and processing, and in this section, the preparation of the dataset is going to be described step by step. Essentially, the dataset was imported to a *Jupyter* notebook as a dataframe, the categorical variables were encoded as numerical ones, and the data was scaled so that it didn't bias the importance of each feature.

Firstly, the NSL-KDD dataset, as a set of .csv files (*KDDTrain+ and KDDTest+*), was loaded into the notebook by using the *pandas* library. Pandas is a crucial library for most of the operations done on the data, from reading/writing, to handling the dataset column by column and encoding it.

Using pandas, the dataset was loaded into two *dataframe* type variables, one for the training and one for the testing subset. Their lengths are $125{,}973$ and $22{,}544$ records respectively, and they both have 43 columns $(0-42)$.

Other than the two multiclass datasets *(Figure 18)*, two more pairs of training – test dataframes were created, one for binary classification *(Figure 19),* where #42 labels were turned into 'normal' and 'abnormal', and one for the 4-class classification *(Figure* 20*)*, where #42 labels were turned into 'normal', 'DoS', 'Probe', 'R2L' and 'U2R' labels.

Below, are snapshots of the heads (first five rows) of all the dataframes, as they are displayed in the Jupyter notebook:

df_nsltrain:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.17	0.03	0.17	0.00	0.00	0.00	0.05	0.00	normal	20
1	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88	0.00	0.00	0.00	0.00	0.00	normal	15
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00	0.00	1.00	1.00	0.00	0.00	neptune	19
3	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03	0.04	0.03	0.01	0.00	0.01	normal	21
4	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	normal	21

df_nsltest:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	private	REJ	0	0	0	0	0	0	...	0.04	0.06	0.00	0.00	0.0	0.0	1.00	1.00	neptune	21
1	0	tcp	private	REJ	0	0	0	0	0	0	...	0.00	0.06	0.00	0.00	0.0	0.0	1.00	1.00	neptune	21
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	0.61	0.04	0.61	0.02	0.0	0.0	0.00	0.00	normal	21
3	0	icmp	eco_i	SF	20	0	0	0	0	0	...	1.00	0.00	1.00	0.28	0.0	0.0	0.00	0.00	saint	15
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	0.31	0.17	0.03	0.02	0.0	0.0	0.83	0.71	mscan	11

Figure 18: Multiclass training and test dataframes (heads)

df_nsltrain_binclas:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.17	0.03	0.17	0.00	0.00	0.00	0.05	0.00	normal	20
1	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88	0.00	0.00	0.00	0.00	0.00	normal	15
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00	0.00	1.00	1.00	0.00	0.00	abnormal	19
3	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03	0.04	0.03	0.01	0.00	0.01	normal	21
4	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	normal	21

df_nsltest_binclas:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	private	REJ	0	0	0	0	0	0	...	0.04	0.06	0.00	0.00	0.0	0.0	1.00	1.00	abnormal	21
1	0	tcp	private	REJ	0	0	0	0	0	0	...	0.00	0.06	0.00	0.00	0.0	0.0	1.00	1.00	abnormal	21
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	0.61	0.04	0.61	0.02	0.0	0.0	0.00	0.00	normal	21
3	0	icmp	eco_i	SF	20	0	0	0	0	0	...	1.00	0.00	1.00	0.28	0.0	0.0	0.00	0.00	abnormal	15
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	0.31	0.17	0.03	0.02	0.0	0.0	0.83	0.71	abnormal	11

Figure 19: Binary training and test dataframes (heads)

df_nsltrain_4clas:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.17	0.03	0.17	0.00	0.00	0.00	0.05	0.00	normal	20
1	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88	0.00	0.00	0.00	0.00	0.00	normal	15
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00	0.00	1.00	1.00	0.00	0.00	DoS	19
3	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03	0.04	0.03	0.01	0.00	0.01	normal	21
4	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	normal	21

df_nsltest_4clas:

	0	1	2	3	4	5	6	7	8	9	...	33	34	35	36	37	38	39	40	41	42
0	0	tcp	private	REJ	0	0	0	0	0	0	...	0.04	0.06	0.00	0.00	0.0	0.0	1.00	1.00	DoS	21
1	0	tcp	private	REJ	0	0	0	0	0	0	...	0.00	0.06	0.00	0.00	0.0	0.0	1.00	1.00	DoS	21
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	0.61	0.04	0.61	0.02	0.0	0.0	0.00	0.00	normal	21
3	0	icmp	eco_i	SF	20	0	0	0	0	0	...	1.00	0.00	1.00	0.28	0.0	0.0	0.00	0.00	Probe	15
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	0.31	0.17	0.03	0.02	0.0	0.0	0.83	0.71	Probe	11

Figure 20: 4-class training and test dataframes (heads)

The difference between each pair of dataframes can be seen in column #41, where the traffic type label has a different set of values.

The next step in the process is to clean any data with the wrong format, drop records with missing values or redundant records. As was explained in a previous section, one of the things that were upgraded from the KDD dataset is that all redundant data were deleted, so the dataset has only unique records of traffic. After checking that there are no missing features, no missing values, and no wrong formatted values in the dataset, some adjustments were made to the dataframes. Firstly, the last column of the dataset, which is the difficulty level of the records, was dropped, and saved separately into two lists, one for the training and one for the test difficulty levels. The distributions of each difficulty list can be found in *Figure* 21 and *Figure* 22, where we can see that most records have the highest score (21/21):

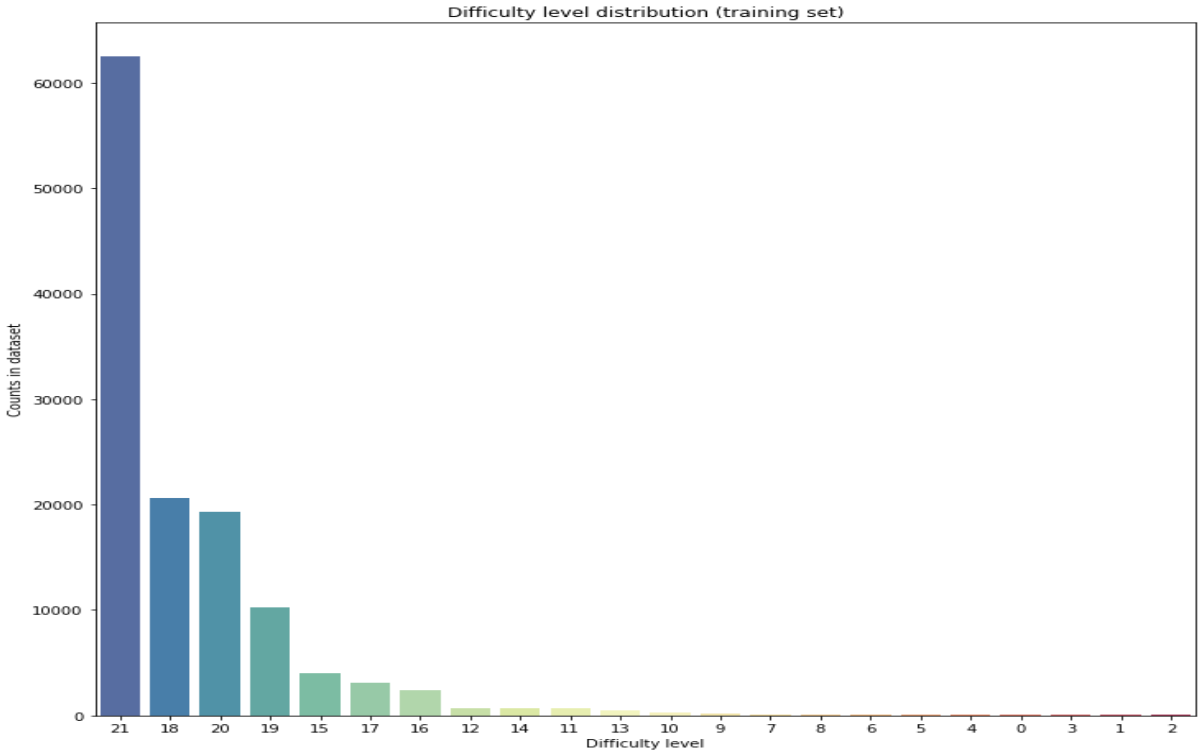


Figure 21: difficulty distribution in training set

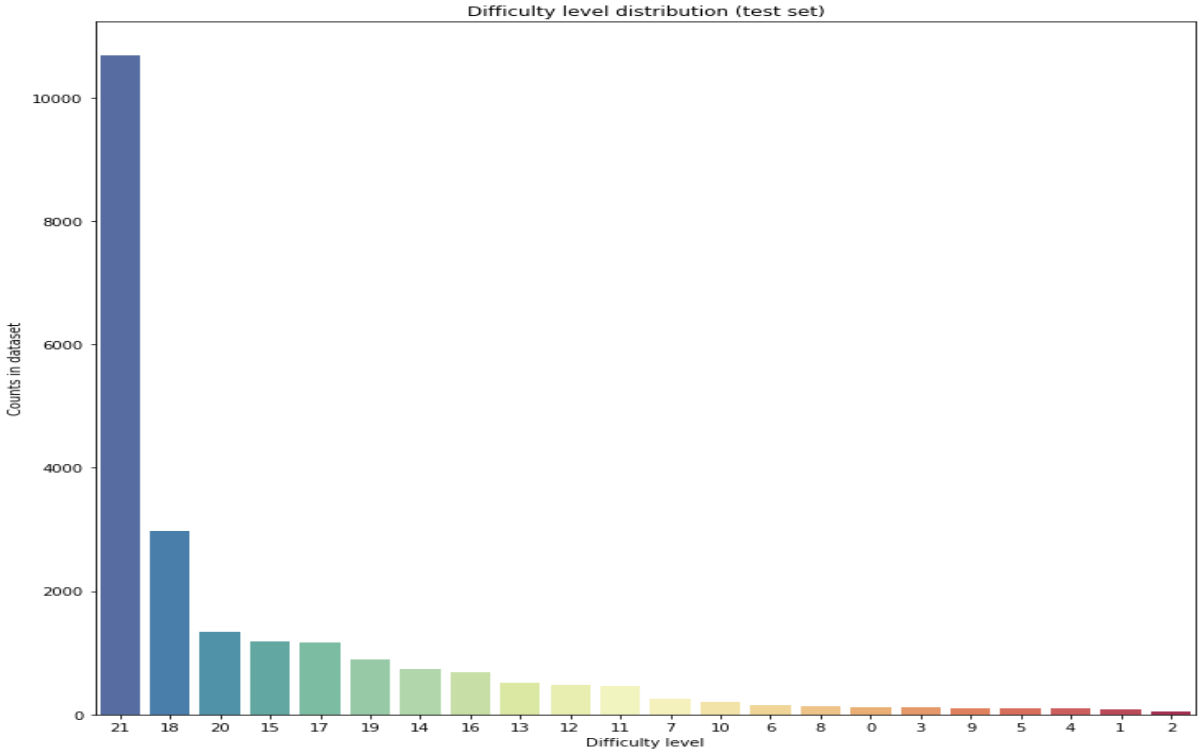


Figure 22: difficulty distribution in test set

Specifically, in the training set there are $62{,}557$ records (out of $125{,}973$), or 49.66%, and in the test set $10{,}694$ (out of $22{,}544$ records), which amount to 47.44%. Difficulty level $18/21$ follows in both subsets, with a presence of $20{,}667$ (16.41%) and $2{,}967$ (13.16%) respectively.

After separating column $\#42$, one more was deleted from all the dataframes, column $\#19$ (number of outbound commands in an ftp session). This column, which was all zeros, became NaN during the correlation calculations, and also, it didn't seem to offer any information to the dataset.

4.3.1. One-hot encoding

After cleaning the data, the categorical variables need to be changed into numerical, in order to be counted in the process of correlation calculations and then to be fed into the model. As described before, there are four features in the dataset that represent classification: protocols, services, flags, and attack types *(see 4.2.1. Categorical features).* While the first three are common in all the dataframes created, the attack type makes it necessary to create different encoded representations for each dataframe.

The way that the categorical features were encoded into numerical variables is through *one-hot encoding* [27]. With one-hot encoding, specifically by creating *dummy* variables out of the categorical ones, one label is turned into a vector of N-dimensions, where N is the number of all the different values this categorical variable might have.

For example, in column $\#1$ are the protocols used for each record. In the dataset, there are three protocol categories: $\{TCP, UDP, ICMP\}$, and each record can have only one of these values. The protocols, like all other categorical features in this dataset, do not have a hierarchical relationship with each other, which means that they can't be replaced by integer values, like: $\{0, 1, 2\}$, as they would indicate an order to the different categories. With one-hot encoding of the protocols, the categorical variables are turned into 3-dimensional vectors: $\{[1, 0, 0], [0, 1, 0], [0, 0, 1]\}$. TCP is represented by $[1, 0, 0]$, similarly UDP with $[0, 1, 0]$ and $ICMP$ with $[0, 0, 1]$. As the name on-hot encoding suggest, each record must have all dimensions zero, except for the one that represents its category.

With the *.get_dummies* method, the categorical features are all moved to the end of the dataframe, and the one-hot encoded vectors are expanded as different columns. This is shown in *Table 5*, where columns 0 and $4 - 40$ are ordered the same as before, while columns 1 (protocols), 2 (services), and 3 (flags) have moved to the tail of the dataframe, and the previously one-column-each feature has expanded to $3, 70$ and 11 columns respectively (in the training set).

In the case of column 41 (type of traffic), it has expanded to different numbers of labels, according to the classification. There are 23 traffic labels for the multiclass training set (in the respective test set, there are 38 different labels), 2 labels for the binary classification and 5 labels for the four-class classification training and test sets.

One-hot encoding has a minor flow, which can cause a problem if not considered. As it was mentioned, the number of traffic labels differs between the training set and the test set, resulting in a different number of columns after the encoding. The same thing happens in the

case of the services feature (col. #2), where, in the training set we can find 70 different services, and in the test set we find 64. This problem was later addressed, at the later steps of the pre-processing, as the subsets were prepared to be fed into the models.

Table 5: labels of the dataframes before and after one-hot encoding

Labels of columns before	Labels of columns in multiclass training dataframe, after one-hot encoding	Labels of columns in binary classification training dataframe, after one-hot encoding	Labels of columns in 4-class classification training dataframe, after one-hot encoding
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]	[0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, '1_icmp', '1_tcp', '1_udp', '2_IRC', '2_X11', '2_Z39_50', '2_aol', '2_auth', '2_bgp', '2_courier', '2_csnet_ns', '2_ctf', '2_daytime', '2_discard', '2_domain', '2_domain_u', '2_echo', '2_eco_i', '2_ecr_i', '2_efs', '2_exec', '2_finger', '2_ftp', '2_ftp_data', '2_gopher', '2_harvest', '2_hostnames', '2_http', '2_http_2784', '2_http_443', '2_http_8001', '2_imap4', '2_iso_tsap', '2_klogin', '2_kshell', '2_ldap', '2_link', '2_login', '2_mtp', '2_name', '2_netbios_dgm', '2_netbios_ns', '2_netbios_ssn', '2_netstat', '2_nnsp', '2_nntp', '2_ntp_u', '2_other', '2_pm_dump', '2_pop_2', '2_pop_3', '2_printer', '2_private', '2_red_i', '2_remote_job', '2_rje', '2_shell', '2_smtp', '2_sql_net', '2_ssh', '2_sunrpc', '2_supdup', '2_systat', '2_telnet', '2_tftp_u', '2_tim_i', '2_time', '2_urh_i', '2_urp_i', '2_uucp', '2_uucp_path', '2_vmnet', '2_whois', '3_OTH', '3_REJ', '3_RSTO', '3_RSTOS0', '3_RSTR', '3_S0', '3_S1', '3_S2', '3_S3', '3_SF', '3_SH', '41_back', '41_buffer_overflow', '41_ftp_write', '41_guess_passwd', '41_imap', '41_ipsweep', '41_land', '41_loadmodule', '41_multihop', '41_neptune', '41_nmap', '41_normal', '41_perl', '41_phf', '41_pod', '41_portsweep', '41_rootkit', '41_satan', '41_smurf', '41_spy', '41_teardrop', '41_warezclient', '41_warezmaster']	[0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, '1_icmp', '1_tcp', '1_udp', '2_IRC', '2_X11', '2_Z39_50', '2_aol', '2_auth', '2_bgp', '2_courier', '2_csnet_ns', '2_ctf', '2_daytime', '2_discard', '2_domain', '2_domain_u', '2_echo', '2_eco_i', '2_ecr_i', '2_efs', '2_exec', '2_finger', '2_ftp', '2_ftp_data', '2_gopher', '2_harvest', '2_hostnames', '2_http', '2_http_2784', '2_http_443', '2_http_8001', '2_imap4', '2_iso_tsap', '2_klogin', '2_kshell', '2_ldap', '2_link', '2_login', '2_mtp', '2_name', '2_netbios_dgm', '2_netbios_ns', '2_netbios_ssn', '2_netstat', '2_nnsp', '2_nntp', '2_ntp_u', '2_other', '2_pm_dump', '2_pop_2', '2_pop_3', '2_printer', '2_private', '2_red_i', '2_remote_job', '2_rje', '2_shell', '2_smtp', '2_sql_net', '2_ssh', '2_sunrpc', '2_supdup', '2_systat', '2_telnet', '2_tftp_u', '2_tim_i', '2_time', '2_urh_i', '2_urp_i', '2_uucp', '2_uucp_path', '2_vmnet', '2_whois', '3_OTH', '3_REJ', '3_RSTO', '3_RSTOS0', '3_RSTR', '3_S0', '3_S1', '3_S2', '3_S3', '3_SF', '3_SH', '41_abnormal', '41_normal']	[0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, '1_icmp', '1_tcp', '1_udp', '2_IRC', '2_X11', '2_Z39_50', '2_aol', '2_auth', '2_bgp', '2_courier', '2_csnet_ns', '2_ctf', '2_daytime', '2_discard', '2_domain', '2_domain_u', '2_echo', '2_eco_i', '2_ecr_i', '2_efs', '2_exec', '2_finger', '2_ftp', '2_ftp_data', '2_gopher', '2_harvest', '2_hostnames', '2_http', '2_http_2784', '2_http_443', '2_http_8001', '2_imap4', '2_iso_tsap', '2_klogin', '2_kshell', '2_ldap', '2_link', '2_login', '2_mtp', '2_name', '2_netbios_dgm', '2_netbios_ns', '2_netbios_ssn', '2_netstat', '2_nnsp', '2_nntp', '2_ntp_u', '2_other', '2_pm_dump', '2_pop_2', '2_pop_3', '2_printer', '2_private', '2_red_i', '2_remote_job', '2_rje', '2_shell', '2_smtp', '2_sql_net', '2_ssh', '2_sunrpc', '2_supdup', '2_systat', '2_telnet', '2_tftp_u', '2_tim_i', '2_time', '2_urh_i', '2_urp_i', '2_uucp', '2_uucp_path', '2_vmnet', '2_whois', '3_OTH', '3_REJ', '3_RSTO', '3_RSTOS0', '3_RSTR', '3_S0', '3_S1', '3_S2', '3_S3', '3_SF', '3_SH', '41_DoS', '41_Probe', '41_R2L', '41_U2R', '41_normal']
In total: 42	In total: 144	In total: 123	In total: 126

4.3.2. Correlation

The encoded dataframe is now ready to have its correlation measured, since all the variables are going to be taken into account. The pandas *.corr* function [28] is used to find the pair wise correlation of all the columns in the dataset, thus the relationship between the features is explored. If the categorical variables were not changed to numerical, they wouldn't have been considered during the calculations, so even the most interesting label, that of the type of traffic, which is the goal of the model, wouldn't be correlated with the features of the dataset. Correlation was calculated for all the dataframes, so there are six figures in total, for multiclass *(Figure* **23***, Figure* **24***),* binary *(Figure* **25***, Figure 26)* and 4-class *(Figure* **27***, Figure 28)* classification.

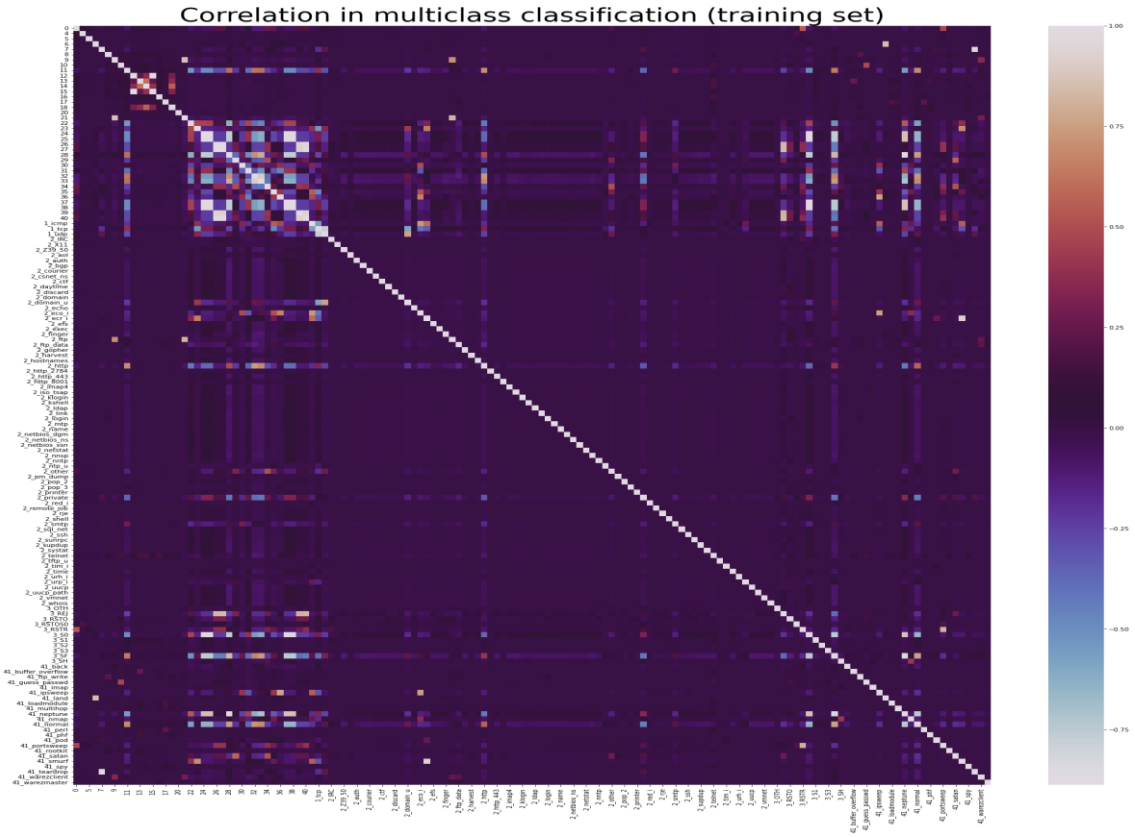


Figure 23: correlation in multiclass training set

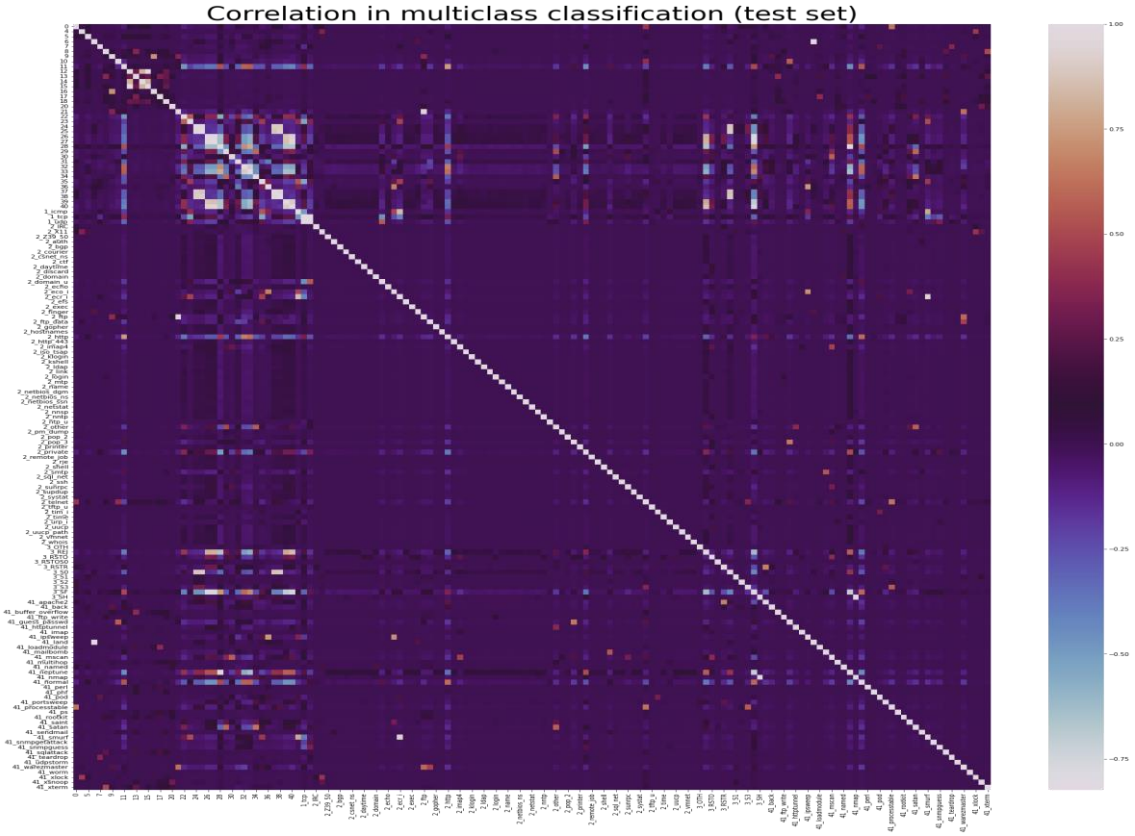


Figure 24: correlation in multiclass test set

Figure 25: correlation in binary training set

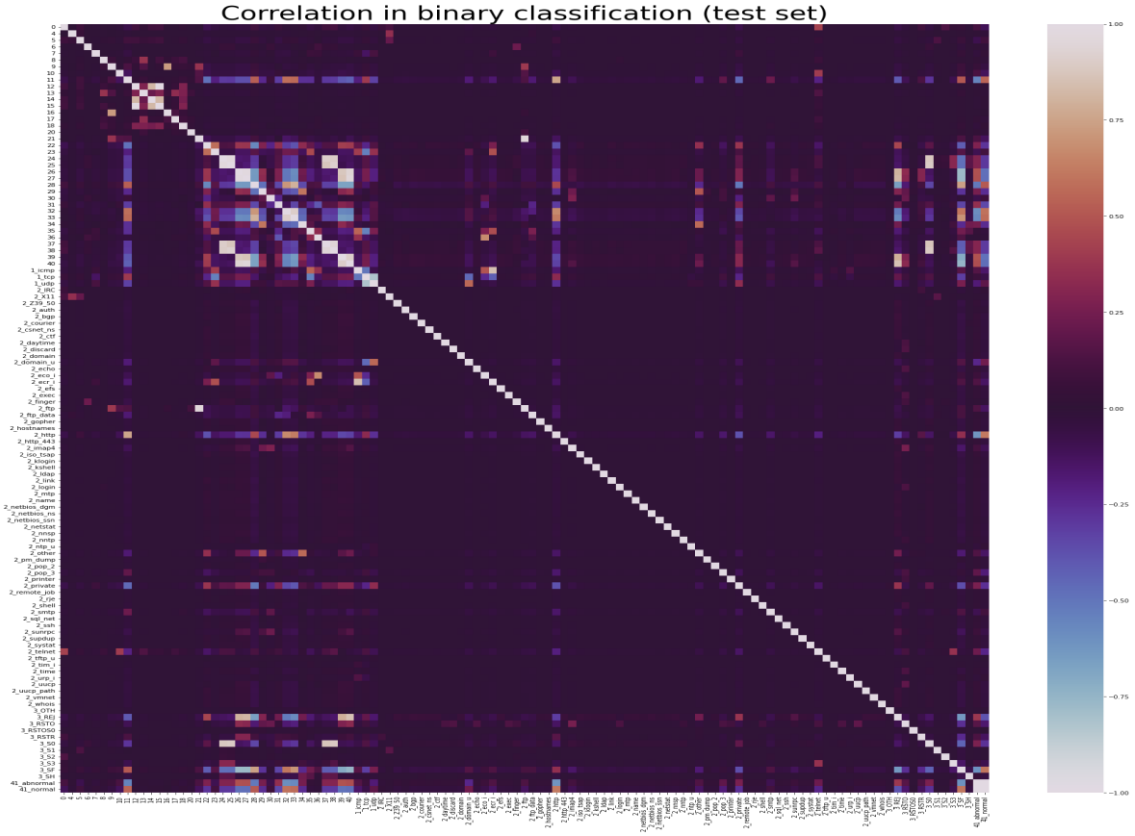


Figure 26: correlation in binary test set

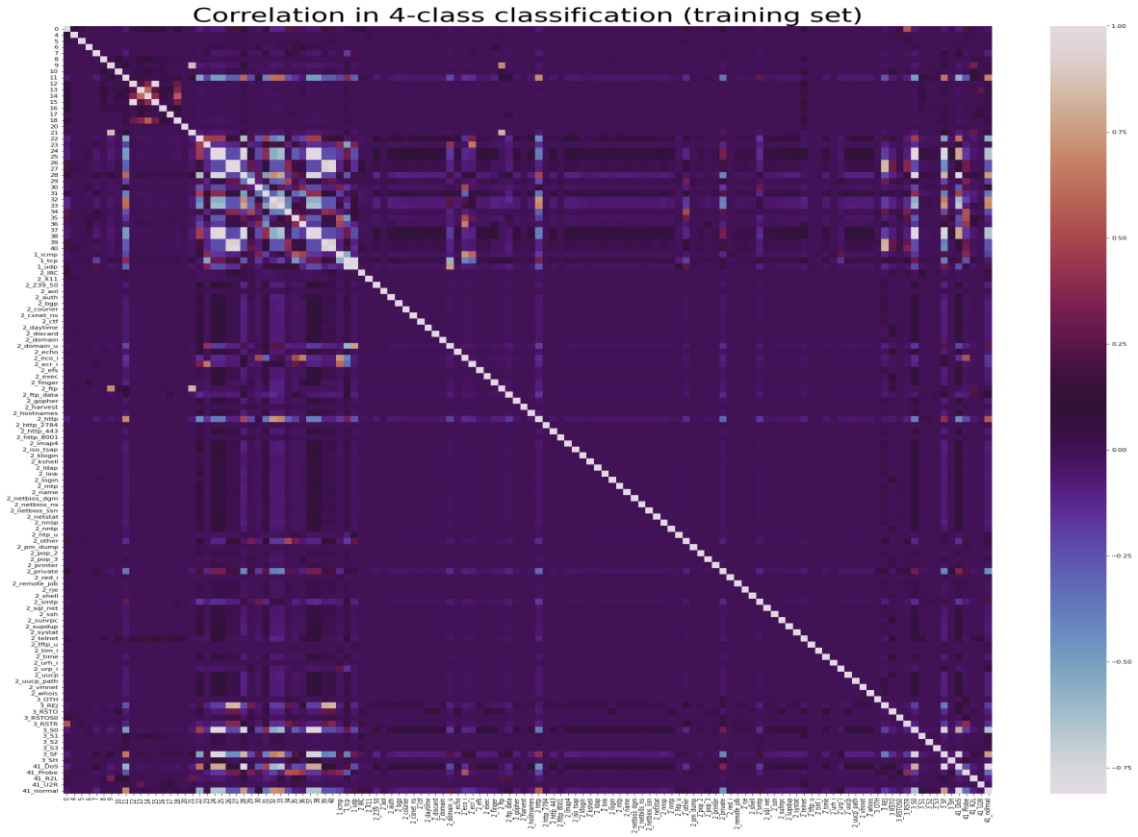


Figure 27: correlation in 4-class training set

Figure 28: correlation in 4-class test set

The correlation ranges from -1 to $+1$. When it trends towards $+1$, it shows that the two columns have a proportional relationship (when $A \uparrow$ then $B \uparrow$). On the other hand, when correlation is closer to -1, then the two columns have an inversed proportional relationship to each other (when $A \uparrow$ then $B \downarrow$). When two features have no relationship to each other, so the way one changes doesn't influence the other, correlation is 0. For the visualisation of following graphs, a palette was chosen that highlights correlation the closer it is to $+1$ and -1, while correlations close to 0 are dark.

The correlation calculations are done pairwise, which means that the product of this metric is a matrix with dimensions $\#\ columns\ \times \#\ columns$. This is why the diagonal highlighted in all the graphs has correlation equal to 1, it calculates correlation between the column and itself. Because of the difference in the resulting columns due to one-hot encoding, the correlation matrices also have different sizes from each other, as is shown in *Table 6* below:

Table 6: correlation matrices dimensions

Multiclass training set	144×144
Multiclass test set	153×153
Binary class training set	123×123
Binary test set	117×117
4-class training set	126×126
4-class test set	120×120

Using the *seaborn* python library, the matrices were visualised as heatmaps, where high correlation (towards $+1$ and -1) has lighter hues the more it tends to ± 1, and darker the more it nears 0.

It is clear and expected (since all the features apart from the traffic type are the same) that all the dataframes behave the same way, with higher intercorrelation among the time-based and host-based features (columns $\#21 - \#40$). Those are the features that also seem to be influenced by the categorical features the most, as we can see in the bottom left of the heatmaps. Another interesting thing we can see from the correlation heatmaps is that in the multiclass set *(Figure 23, Figure 24)* in the last columns, where the encoded traffic types are, the correlation is lower than in both the binary and the 4-class classifications; this is expected, when the types of classifications are compared. What is interesting in the multiclass case, is that the attacks show no correlation to each other, be it attacks of the same category or other classes. In fact, the correlation between different classes of attacks in the 4-class classification *(Figure 277, Figure 288)* seems to be higher than that of the multiclass separated attacks that belong in the same class.

Thanks to the correlation between the features, especially the correlation between the traffic type and the other features of each record, we can understand what makes the model classify something as a DoS or a U2R attack, or as normal traffic.

4.3.3. X and Y components, scaling the data

With the correlation of the NSL-KDD calculated, we saw the connection between the target labels and the features of the dataset in each classification scenario, which features mostly affect the prediction of the model. Now, the next step is to split the dataframes' features (columns $0-40$) from the target labels (column 41), into X and Y components, so that they can be used as input and output respectively for the models we use.

New instances of the dataframes are created by copying the first 40 columns of the original training and test dataframes into x-train/x-test dataframe type variables, and the last column ($\#41$) into y-train/y-test variables (columns $\#19$ and $\#42$ are deleted). This is repeated for all the different classification cases, even though the X component is essentially the same in all of them. These variables are created from the dataframes as they were before one-hot encoding, as it is easier to load them without minding the amount of encoded extra columns that are created after it; thus, it is necessary to do the encoding again, but only on the X component (columns $1, 2$ and 3), leaving the Y dataframe of traffic type labels a categorical list of one column.

Next is the alignment of the training and test arrays, because the one-hot encoded training set is a 125973×121 (records × columns) array, while the test is 22544×115. As they are, the two arrays cannot be used by the same model; they must be uniform, as the structure of the model recognises a specific form of input.

As was mentioned above, due to the one-hot encoding algorithm used, the categorical columns move at the end of the features space and are expanded to all their unique labels as $1s$ and $0s$. It is important to consider how the alignment occurs in a way that doesn't disturb the classes of each previously categorical variable. With the pandas method *.align()* it was possible to add the extra columns at the right place (with the '*outer' join* option) and add the value 0 to them (with the *fill value* option), so that the extra categories not found in one of the dataframes was added in between the rest of the particular categorical feature and conformed to the one-hot way of encoding the variables. As a result, all the datasets were 125973×121 or 22544×121, and there was no NaN value in any of them.

After the alignment of the training and test datasets, the dataframes were scaled, using the *Standard Scaler* from the *sklearn.preprocessing* library [29]. Scaling the numerical data is an important step in pre-processing, as their different ranges and dimensions result in bias when the weights of the model are calculated. The range of the different features, seen in *Annex A (Table* 9*)* is $0 - 1{,}379{,}963{,}888$ in the case of column 4, but only $0/1$ in all the binary variables, or even hundredth decimals in the float variables that represent rates. No matter how few the datapoints with values this far apart are, the model needs to be trained and validated on similar sizes of data, so that the relationships between each record and each feature can be better recognised.

Standard scaler follows the standard normal distribution to calculate the value of each datapoint, which means that it takes mean $= 0$ and scales the data so that the total variance, meaning the new range of the data, is $= 1$ (unit variance). The scaling is calculated as:

$$x' = \frac{x - \mu}{s}$$

Equation 6: standard scaling equation

Where x' is the new scaled data, x is the data to be scaled, μ is the mean of the training samples and s is the standard deviation.

Standard scaling occurs in two steps for the training data, the fitting phase, and the transformation phase. The *fit(data)* function is used to compute the mean and standard deviation of each (numeric) feature, while *transform(data)* is used after the fitting to perform the scaling of the data, using the variables calculated with the *fit* function. This way, the scaler is trained (calculates μ, s) on the training set, and then, with those parameters set, the transformation is also applied on the test data. Thus, it is important to apply *fit* and *transform* on the training data, but only *transform* on the test data, so that the model is not biased with information from the test data. While theoretically, the training and test set might have mean and deviation values that are very close, we shouldn't let the model be influenced by the test set distribution and features when it is training. Also, the test data should be scaled according to the training set's distribution parameters, so that its divergence from the original training data is prominent and the model truly tested on unknown records.

After aligning and scaling the data, the dataset looks like this:

$$\begin{bmatrix}
[-0.11024922 & -0.0076786 & -0.00491864 \ldots -0.01972622 & 0.82515007 & -0.04643159] \\
[-0.11024922 & -0.00773737 & -0.00491864 \ldots -0.01972622 & 0.82515007 & -0.04643159] \\
[-0.11024922 & -0.00776224 & -0.00491864 \ldots -0.01972622 & -1.21190076 & -0.04643159] \\
\multicolumn{5}{c}{\ldots} \\
[-0.11024922 & -0.00738219 & -0.00482315 \ldots -0.01972622 & 0.82515007 & -0.04643159] \\
[-0.11024922 & -0.00776224 & -0.00491864 \ldots -0.01972622 & -1.21190076 & -0.04643159] \\
[-0.11024922 & -0.00773652 & -0.00491864 \ldots -0.01972622 & 0.82515007 & -0.04643159]]
\end{bmatrix}$$

Figure 29: training dataset (multiclass) after standard scaling

Now that the scaling of the data is finished, the training and test datasets are ready to be fed into models for the three different classification scenarios (multiclass, binary, and 4-classes classification), for the intrusion detection performance to be measured.

To sum up, this section describes the whole pre-processing phase of this project. We saw how the data was loaded into dataframes and how the three kinds of classifications were created, by changing the traffic type labels in column $\#41$ into 'normal' and 'abnormal' in the case of binary classification, or 'normal', 'DoS', 'Probe', 'U2R' and 'R2L' types of attacks in the 4 attack classes. Then, the distribution of the categorical features, that describe the type of traffic, was analysed, and the categorical variables of the dataframes were encoded into numerical values, via one-hot encoding. Correlation calculations showed that the features that mostly affect the type of traffic are the time- and host-based ones. After that, the datasets were split into X and

Y components, so that they are ready to be fed into the models, the X components were once again one-hot encoded into numerical variables, and the training and test sets were aligned to each other and scaled to unit variance using the standard scaler.

In the next section, the classification models are going to be tested out and evaluated on their performance on the dataset.

5. Evaluation and results

The training and test datasets, which were pre-processed earlier *(section 4.3. Pre-processing of the NSL-KDD dataset)* showed that they were very different from each other, especially when it came to the distribution of the traffic type labels *(Table 1)* and of the difficulty levels *(Figure 21, Figure* 22*22)*. This divergence led to interesting results in the experimental phase of this project, where the models described above *(Classification models analysis)* were put to the test. It was observed that the test dataset, on which predictions were made, showed far lower accuracy scores than the accuracy acquired during training. Naturally, overfitting was a problem that was first addressed, but still the models seemed to stabilise at the performance shown below, in *Table 7.* For a deeper view into the performance of the models and the effect of the differences between the *KDDTrain+* and *KDDTest+* datasets, two cases were created: in *case A,* the five models were applied to all the classification scenarios (multiclass, binary, 4-class), using the *KDDTrain+* as training dataset, and the *KDDTest+* as test (validation) dataset, as they were prepared during the pre-processing phase *(4.3. Pre-processing of the NSL-KDD dataset).* In *case B*, the same models were used (mostly with the same parameters that optimized their performance) on all the classifications, but here only the *KDDTrain+* was used as training and test set, by splitting it with the commonly used *train_test_split* [30] utility from the *sklearn* library, after being pre-processed like before.

When the same algorithms were applied to *case B,* it showed that the models were working exceptionally, as is shown in *Table 8*. Due to the very homogenous distribution of the training and validation parts of the dataset, the performance both in training and testing phases is the same and gets very high results.

Table 7: summary/comparison of classification algorithms performance in case A

	CLASSIFICATION ALGORITHM	CLASS SCENARIO	TRAINING SET	TEST SET
CASE A: using the *KDDTrain+* and *KDDTest+* as training and test sets	**LOGISTIC REGRESSION**	multi	0,99	0,70
		binary	0,97	0,75
		4-class	0,99	0,76
	DECISION TREE	multi	1,00	0,71
		binary	1,00	0,79
		4-class	1,00	0,76
	K NEAREST NEIBOURS	multi	0,99	0,72
		binary	0,99	0,77
		4-class	0,99	0,74
	GAUSSIAN NAÏVE BAYES	multi	0,77	0,53
		binary	0,84	0,55
		4-class	0,65	0,42
	MULTI LAYER PERCEPTRON	multi	1,00	0,72
		binary	1,00	0,79
		4-class	1,00	0,77

Table 8: summary/comparison of classification algorithms performance in case B

	CLASSIFICATION ALGORITHM	CLASS SCENARIO	TRAINING SET	TEST SET
CASE B: splitting the *KDDTrain+* in training and test/validation subsets	**LOGISTIC REGRESSION**	multi	0,99	0,99
		binary	0,97	0,97
		4-class	0,99	0,99
	DECISION TREE	multi	1,00	1,00
		binary	1,00	1,00
		4-class	1,00	1,00
	K NEAREST NEIBOURS	multi	0,99	0,99
		binary	0,99	0,99
		4-class	0,99	0,99
	GAUSSIAN NAÏVE BAYES	multi	0,76	0,76
		binary	0,85	0,85
		4-class	0,65	0,65
	MULTI LAYER PERCEPTRON	multi	1,00	1,00
		binary	1,00	1,00
		4-class	1,00	1,00

Below *(Figure* 3030*, Figure 31)*, we can see the accuracy scores of *Table 7* and *Table 8* in a diagram form, where it is easier to see the difference between the two cases (using KDDTrain+ and KDDTest+ versus splitting the KDDTrain+ dataset into training and test sets), but what is also noticeable is the similarity in the behaviour of all the models between training and test accuracy scores.

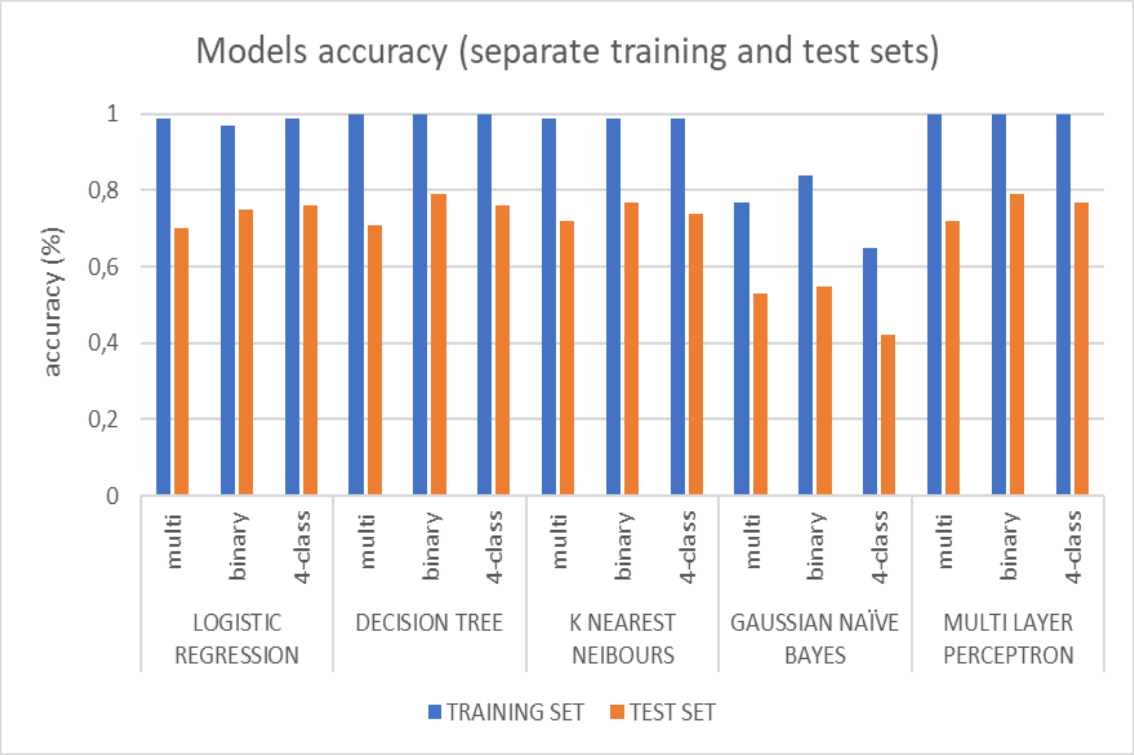


Figure 30: accuracy scores of all models and classification scenarios for case A

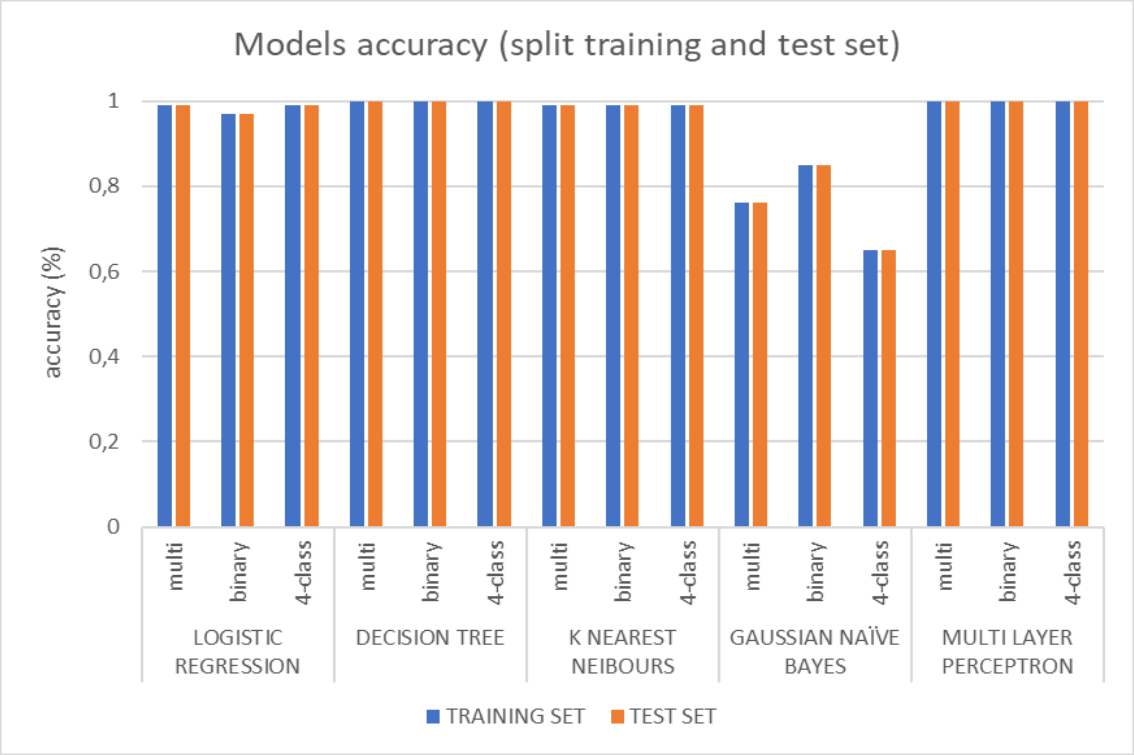


Figure 31: accuracy scores of all models and classification scenarios for case B

It is important to emphasize that these are the final scores that the models obtained, as they trended towards that value and stabalized there afterwards, with no more fine-tuning happening, and those are the values that are the closest between the training and test sets.

5.1. Interpreting the Classification Reports

The classification reports, found in *Annex C: list of all the classification reports*, contain all the information extracted from the model analysis, for the parameters that showed optimized results. The report displays four columns of information, "precision", "recall", "f1 score" and "support". Since most of the models work in a one-against-all way ($y = 1$ -positive- if it is the class we are looking for, $y = 0$ -negative- if it is any other class), there are four possible outcomes of the algorithm calculation:

- True positive *(TP)*: the entry was positive, and the model predicted positive.
- False positive *(FP)*: the entry was negative, and the model predicted positive.
- False negative *(FN)*: the entry was positive, and the model predicted negative.
- True negative *(TN)*: the entry was negative, and the model predicted negative.

These four percentages that make up the model's performance for each label, will be used to create the three metrics of performance for the classification report [31][32][33].

Precision (Equation 7) is the measure of how accurate the model's positive predictions are, how much it's able to avoid wrongly labelling something as positive:

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

Equation 7: precision equation

Recall (Equation 8) is the ability of the model to find all the positive values in the dataset for this instance:

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

Equation 8: recall equation

F1-score (Equation 9) is the harmonic mean of precision and recall, and is a measure of the model's accuracy, for the classification of each instance:

$$F_1 = \frac{2}{precision^{-1} + recall^{-1}} = \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Equation 9: F1-score - harmonic mean equation

Lastly, *Support* is simply the times that each specific class is encountered in the dataset, the instances of each label. It is from those instances that the precision and recall are calculated for each label of the dataset, in a binary way (one-against-all).

The average values of the report below the label-by-label metrics lead the results from binary to multiclass classification [34]. Most importantly, *accuracy (Equation 11,Equation 11)* measures the overall ability of the model to classify correctly over all of the values. If $\hat{y}_i$ is the predicted value of sample i and y_i is the real value, then accuracy is defined as:

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

Equation 10: accuracy equation

Or more intuitively:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{correct\ classifications}{all\ classifications}$$

Equation 11: intuitive accuracy equation

Macro average is simply the mean of all the above binary metrics, for precision, recall and f-1 score respectively, taking all classes as of equal importance, which is often untrue, especially in unbalanced datasets such as NSL-KDD; *weighted average* is the mean value of the binary metrics, with each class's score weighted by its presence in the dataset (the support value). The *weighted average* is much closer to the accuracy score, as we can see in the classification

reports of our models, because of the imbalance of our dataset, and shows a quick evaluation of the performance of each binary classification as a whole.

5.2. Evaluation and results compared to relevant research

In reality, *case B* is not very useful, because there is rarely any chance to encounter traffic data so close to the training data of the model, especially with the rapid rate that network exploitations evolve today; it was mostly done to test if there was an overfitting problem, as validation for the training phase of the models, and to apply the same practices that are usually done step-by-step in most machine learning projects, which usually split the original dataset into training and test subsets.

The test set of the NSL-KDD, with its difference in the distribution of the labels , services, flags and many more features, reflects more of the real world, and the performance of all the models actually reaches almost the same levels as some of the latest research, even much more complex and innovative models, like [35], [36], [12] and [9].

In [35], the best results of all are found, with a record 89% accuracy reached in a LSTM model. Except that, they use a Deep CNN, combined with Denoising and Contractive AE in different balances, and reach $81 - 85\%$. Using more classical approaches similar to ours, (kNN, DT, MLP, RF) they reach $74 - 82\%$ accuracy. [12] have the second best accuracy results, with their implementation being an AE, followed by another sparse AE network, and for the output layer they have put a LR classifier, that only provides binary classificaction. With these, they reach 87.2% accuracy. In [36], the input goes through multiple CNNs, a BLSTM and an attention layer, in order to reach 84.2% accuracy. With traditional approaches (DT, MLP, RF), they reqach $72 - 78\%$. Lastly, in [9], they developed similar classifiers (DT, DNN) that reached $76 - 79\%$, and with PCA they reduced the features to 6, making accuracy drop to $71 - 75\%$.

[13] and [11] have made studies that are very similar to our own, but they use the whole NSL-KDD dataset as one, and after the preprocessing phase, they split it into training and validation/test subsets, like in our *case B*. The $99 - 99.6\%$ accuracy obtained there in all the classifiers tested looks like the results extracted from our models, found in *Table 8*. However, the problem here lies with the real-life experience that *case B*-like experiments don't provide.

Many similar projects can also be found in Github, since the NSL-KDD is a very popular dataset for intrusion detection, some of which only utilise the *KDDTrain+_20Percent* and *KDDTest+_20Percent* for easier and faster processing. Most such projects either have minimal optimisation, and mostly analyse the NSL-KDD in depth, or develop only one method of a more advanced technique (CNN, Autoencoder) to develop the model itself better.

In the next section, we will discuss the problems and limitations of this project, and future work that could improve the work done.

6. Discussion and future work

Anomaly detection, and network security in general, is facing a lot of challenges. Some notable ones are:

- The rapid development of networks today, which leads to a great increase of novel and unknown attacks, that take advantage of new gaps and services.
- The ever growing reliance of our society on the Internet, where more and more data are generated and handled every year, already barely within our processing capabilities.
- The Internet of Things (IoT), due to which devices of lower level, thus much less processing power and capabilities, are connected to each other, leaving us exposed to new security gaps that could even affect our health, other than our data.
- The unavailability of open network traffic datasets, especially more recent, that could have newer attacks, because of security concerns and competition among service providers, which could refresh the reserch domain.
- The incompetence that unsupervised learning still shows, even though it is very appropriate for anomaly detection, because the performance of models with unlabelled data cannot be tested correctly, while labelling whole datasets is a specially time consuming and difficult process.

Further improvement on this particular project could include two types of upgrades. Firstly, since the NSL-KDD dataset is already labelled, there are many unsupervised learning mechanisms that could then be validated through the respective labels, including attention mechanisms, autoencoders, and clustering methods that we could optimise and compare. Even with supervised learning, a dive into more advanced DNN methodologies would provide better results, and the models could be much more flexible; DNN techniques that could be tested are Convolutions and Pooling methods, RNNs or LSTM approaches, etc. Deep Neural Networks are a central part of the machine learning and AI research nowadays, naturally, because of their flexible architecture, robust performance, and abundance of functions for every part of the models. Using only the normal traffic of the dataset, as is done in [7], could be very useful for unsupervised methods like autoencoders, that learn from the pattern of normal data, and recognise anomalies based on their deviation from them.

The second course of action that could upgrade this project is data centric. With traffic data captured via Wireshark, thanks to the IT department's cooperation, we could use the headers, the only part available to us because of privacy and security concerns, to create the features of the NSL-KDD for the connections provided, or part of them; we have seen throughout the pre-processing phase of our project that some variables influence the data more than others (e.g. with correlation). With the dataset created from the recent traffic data, a whole process of its own to accomplish, we have a potentially worthwhile sample of records, which would be unlabelled. Thankfully, the University has a very secure network infrastructure, being a big campus network where sensitive data moves around, so if we choose a node from the lower levels of it, where the data is already filtered through the security solutions of the network, we

can assume that the vast majority of our data will be normal traffic, without any anomalies present. This kind of data is exceptional to apply to autoencoder models, which learn from the uniform structure of normal data and then recognise the anomalies from their deviation compared to the rest.

A project like that would be good for more advanced research, which can manage both the feature extraction, the data preparation and the advanced unsupervised models development, since there are no labels to apply any supervised methods available. Still, there would be problems and limitations with that kind of development too; namely, there is no way to know if the model would work optimally, as we can't extract the percentages of TP, FP, TN and FN classifications without knowing for sure which datapoints are normal and abnormal traffic, and to test the model would require us to create synthetic attack type of data, or find attack records from available datasets and pre-process them so that they are uniform to our own unlabelled data and tested through the model. This kind of work is too complex and meticulous for the level of a master thesis.

°°°°°

To sum up, despite the challenges and limitations of anomaly detection in the domain of network security, research advances along with machine learning and AI, applying the same innovative methods. Unsupervised learning can be more useful in the real world, since the data in uses don't need to be labelled, and it can detect unknown and novel attacks. In spite of the new wave of research focused on unsupervised learning in the past couple of years, there are still problems and limitations, like mentioned above, and supervised learning, or at least semi/self-supervised learning, still remain the most effective method to study the topic of network security.

Even though the five algorithms are somewhat basic and outdated, this thesis provides results at a satisfactory level, not far behind state-of-the-art experiments. Its benefits lie in the fact that it utilizes one of the most popular datasets available, and goes through a thorough analysis of it, and subsequently compares the performance of five algorithms of supervised learning that are still very commonly used for classification problems and anomaly detection. Even with this kind of approach, we can see that the results of our classifiers are very close to state-of-the-art research, which indicates how useful these methods still are for anomaly detection. There are many ways that the project can be improved in the future, either by developing more advanced models and moving to unsupervised learning solutions, or by trying to create a new dataset with similar features and apply unsupervised methods on it.

Annex A: table of the NSL-KDD features

Table 9: list of all the features in NSL-KDD

#	feature	description	type	value	range
0	duration	time length of the connection	continuous	integer	0-54451
1	protocol type	protocol used in the connection	categorical	string	*NaN*
2	service	destination network service used	categorical	string	*NaN*
3	flag	status of the connection	categorical	string	*NaN*
4	src bytes	number of bytes transferred in a single connection (source to destination)	continuous	integer	0-1379963888
5	dst bytes	number of bytes transferred in a single connection (destination to source)	continuous	integer	0-309937401
6	land	if src. and dst. IP addresses and port numbers are equal then =1, else =0	binary	integer	0 or 1
7	wrong fragment	number of wrong fragments in the connection	discrete	integer	0, 1 or 3
8	urgent	number of urgent packets in the connection (urgent bit activated)	discrete	integer	0-3
9	hot	number of "hot" indicators in the content (entering system dir., creating/executing programs)	continuous	integer	0-101
10	num failed logins	count of failed login attempts	continuous	integer	0-4
11	logged in	if successful login status =1, else =0	binary	integer	0 or 1
12	num compromised	number of compromised conditions	continuous	integer	0-7479
13	root shell	if root shell is obtained =1, else =0	binary	integer	0 or 1
14	su attempted	if "su root" command is attempted or used =1, else =0 (dataset also contains the value 2)	discrete	integer	0, 1 or 2
15	num root	number of "root" accesses or operations performed as a root in the connection	continuous	integer	0-7468
16	num file creations	number of the file creation operations in the connection	continuous	integer	0-100
17	num shells	number of shell prompts	continuous	integer	0-2
18	num access files	number of operations on access control files	continuous	integer	0-9
19	num outbound cmds	number of outbound commands in an ftp session	continuous	integer	0
20	is hot login	if the login belongs to the "hot" list (root/admin) =1, else =0	binary	integer	0 or 1
21	is guest login	if the login is a "guest" =1, else =0	binary	integer	0 or 1

22	count	number of connections to the same destination host as the current connection (in time window)	discrete	integer	0-511
23	srv count	number of connections to the same service (port num.) as the current connection (in time window)	discrete	integer	0-511
24	serror rate	percentage of connections that have activated flags s0, s1, s2 or s3 among the connections in *count (col. 22)*	discrete	float	0.00-1.00
25	srv serror rate	percentage of connections that have activated flags s0, s1, s2 or s3 among the connections in *srv count (col.23)*	discrete	float	0.00-1.00
26	rerror rate	percentage of connections that have activated the flag REJ among the connections in *count (col.22)*	discrete	float	0.00-1.00
27	srv rerror rate	percentage of connections that have activated the flag REJ among the connections in *srv count (col.23)*	discrete	float	0.00-1.00
28	same srv rate	percentage of connections that were to the same service among the connections in *count (col.22)*	discrete	float	0.00-1.00
29	diff srv rate	percentage of connections that were to different services among the connections in *count (col.22)*	discrete	float	0.00-1.00
30	srv diff host rate	percentage of connections that were to different machines among the connections in *srv count (col.23)*	discrete	float	0.00-1.00
31	dst host count	number of connections with the same destination host IP address	discrete	integer	0-255
32	dst host srv count	number of connections with the same port number	discrete	integer	0-255
33	dst host same srv rate	percentage of connections that were to the same services among the connections in *dst host count (col.31)*	discrete	float	0.00-1.00
34	dst host diff srv rate	percentage of connections that were to different services among the connections in *dst host count (col.31)*	discrete	float	0.00-1.00
35	dst host same src port rate	percentage of connections that were to the same services among the connections in *dst host srv count (col.32)*	discrete	float	0.00-1.00
36	dst host srv diff host rate	percentage of connections that were to different destination machines among the connections in *dst host srv count (col.32)*	discrete	float	0.00-1.00

37	dst host serror rate	percentage of connections that have activated the flag s0, s1, s2, or s3 among the connections in *dst host count (col. 31)*	discrete	float	0.00-1.00
38	dst host srv serror rate	percentage of connections that have activated the flag s0, s1, s2, or s3 among the connections in *dst host srv count (col. 32)*	discrete	float	0.00-1.00
39	dst host rerror rate	percentage of connections that have activated the flag REJ among the connections in *dst host count (col.31)*	discrete	float	0.00-1.00
40	dst host srv rerror rate	percentage of connections that have activated the flag REJ among the connections in *dst host srv count (col.32)*	discrete	float	0.00-1.00
41	class	type of traffic classification input	categorical	string	*NaN*
42	difficulty level	difficulty level	discrete	integer	0-21

Annex B: table of all the services in the NSL-KDD dataset

Table 10: list of all the services in the NSL-KDD

service	in training set	in test set		service	in training set	in test set
http	40338	7853		name	451	37
private	21853	4774		mtp	439	32
domain_u	9043	894		echo	434	37
smtp	7313	934		klogin	433	21
ftp_data	6860	851		login	429	29
eco_i	4586	262		ldap	410	19
other	4359	838		netbios_dgm	405	25
ecr_i	3077	752		sunrpc	381	159
telnet	2353	1626		netbios_ssn	362	15
finger	1767	136		netstat	360	26
ftp	1754	692		netbios_ns	347	36
auth	955	67		ssh	311	26
Z39_50	862	45		kshell	299	24
uucp	780	50		nntp	296	21
courier	734	40		pop_3	264	1019
bgp	710	46		sql_net	245	18
whois	693	40		IRC	187	13
uucp_path	689	46		ntp_u	168	10
iso_tsap	687	48		rje	86	8
time	654	36		pop_2	78	13
imap4	647	306		remote_job	78	14
nnsp	630	42		X11	73	15
vmnet	617	43		printer	69	11
urp_i	602	23		shell	65	16
domain	569	51		urh_i	10	0
ctf	563	41		red_i	8	0
csnet_ns	545	34		tim_i	8	6
supdup	544	27		pm_dump	5	16
discard	538	26		tftp_u	3	1
http_443	530	36		aol	2	0
daytime	521	28		harvest	2	0
gopher	518	34		http_8001	2	0
efs	485	33		http_2784	1	0
systat	477	32				
link	475	41				
exec	474	27				
hostnames	460	23				

Annex C: list of all the classification reports

Below, there are two lists, containing all the classification reports that resulted from the analysis of the classification models in the two different use cases. Further information about the models and discussion of their performance can be found in section ***Classification models analysis***, as well as information about the interpretation of the classification reports.

C.1. Case A: using *KDDTrain+ KDDTest+ as training and test sets*

Logistic Regression:

Table 11: logistic regression on the multiclass training set

	precision	recall	f1-score	support
back	0.99	0.97	0.98	974
buffer_overflow	0.60	0.86	0.71	21
ftp_write	0.25	1.00	0.40	2
guess_passwd	0.98	0.96	0.97	54
imap	0.91	1.00	0.95	10
ipsweep	0.97	0.97	0.97	3617
land	0.61	0.85	0.71	13
loadmodule	0.22	0.67	0.33	3
multihop	0.29	0.40	0.33	5
neptune	1.00	1.00	1.00	41223
nmap	0.96	0.93	0.94	1540
normal	1.00	0.99	0.99	67470
perl	0.00	0.00	0.00	0
phf	1.00	1.00	1.00	4
pod	1.00	1.00	1.00	202
portsweep	0.98	1.00	0.99	2890
rootkit	0.20	0.50	0.29	4
satan	0.95	0.98	0.97	3520
smurf	1.00	0.99	0.99	2678
spy	0.00	0.00	0.00	0
teardrop	1.00	1.00	1.00	891
warezclient	0.82	0.88	0.85	833
warezmaster	0.80	0.84	0.82	19
accuracy			**0.99**	**125973**
macro avg	0.72	0.82	0.75	125973
weighted avg	0.99	0.99	0.99	125973

Table 12: logistic regression on the multiclass test set (validation)

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	0
back	0.69	0.33	0.45	748
buffer_overflow	0.05	0.50	0.09	2
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.00	0.17	0.00	6
httptunnel	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	49
ipsweep	0.97	0.76	0.85	181
land	0.43	1.00	0.60	3
loadmodule	0.00	0.00	0.00	2
mailbomb	0.00	0.00	0.00	0
mscan	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	2
named	0.00	0.00	0.00	0
neptune	0.99	0.93	0.96	4984
nmap	0.99	0.41	0.58	174
normal	0.93	0.66	0.77	13724
perl	0.00	0.00	0.00	0
phf	0.50	0.17	0.25	6
pod	0.95	0.71	0.81	55
portsweep	0.92	0.50	0.64	290
processtable	0.00	0.00	0.00	0
ps	0.00	0.00	0.00	0
rootkit	0.08	0.06	0.07	16
saint	0.00	0.00	0.00	0
satan	0.96	0.62	0.75	1151
sendmail	0.00	0.00	0.00	0
smurf	1.00	0.65	0.79	1026
snmpgetattack	0.00	0.00	0.00	0
snmpguess	0.00	0.00	0.00	0
sqlattack	0.00	0.00	0.00	0
teardrop	1.00	0.24	0.39	49
udpstorm	0.00	0.00	0.00	0
warezclient	0.00	0.00	0.00	75
warezmaster	0.00	1.00	0.00	1
worm	0.00	0.00	0.00	0
xlock	0.00	0.00	0.00	0
xsnoop	0.00	0.00	0.00	0
xterm	0.00	0.00	0.00	0
accuracy			**0.70**	**22544**
macro avg	0.27	0.22	0.21	22544
weighted avg	0.94	0.70	0.79	22544

Table 13: logistic regression on the binary training set

	precision	recall	f1-score	support
abnormal	0.97	0.98	0.97	57838
normal	0.98	0.97	0.98	68135
accuracy			**0.97**	**125973**
macro avg	0.97	0.97	0.97	125973
weighted avg	0.97	0.97	0.97	125973

Table 14: logistic regression on the binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.62	0.92	0.74	8713
normal	0.93	0.65	0.76	13831
accuracy			**0.75**	**22544**
macro avg	0.77	0.78	0.75	22544
weighted avg	0.81	0.75	0.76	22544

Table 15: logistic regression on the 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	45984
Probe	0.96	0.98	0.97	11497
R2L	0.80	0.82	0.81	968
U2R	0.54	0.88	0.67	32
normal	0.99	0.99	0.99	67492
accuracy			**0.99**	**125973**
macro avg	0.86	0.93	0.89	125973
weighted avg	0.99	0.99	0.99	125973

Table 16: logistic regression on the 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.84	0.92	0.88	6789
Probe	0.71	0.86	0.78	2005
R2L	0.04	0.50	0.08	238
U2R	0.34	0.74	0.47	31
normal	0.93	0.67	0.78	13481
accuracy			**0.76**	**22544**
macro avg	0.57	0.74	0.60	22544
weighted avg	0.87	0.76	0.80	22544

Decision Tree:

Table 17: decision tree on the multiclass training set

	precision	recall	f1-score	support
back	1.00	1.00	1.00	956
buffer_overflow	1.00	1.00	1.00	30
ftp_write	1.00	1.00	1.00	8
guess_passwd	1.00	1.00	1.00	53
imap	1.00	1.00	1.00	11
ipsweep	1.00	1.00	1.00	3608
land	1.00	0.82	0.90	22
loadmodule	1.00	1.00	1.00	9
multihop	1.00	1.00	1.00	7
neptune	1.00	1.00	1.00	41214
nmap	0.99	1.00	1.00	1485
normal	1.00	1.00	1.00	67342
perl	1.00	1.00	1.00	3
phf	1.00	1.00	1.00	4
pod	1.00	1.00	1.00	200
portsweep	1.00	1.00	1.00	2930
rootkit	0.90	1.00	0.95	9
satan	1.00	1.00	1.00	3632
smurf	1.00	1.00	1.00	2646
spy	1.00	1.00	1.00	2
teardrop	1.00	1.00	1.00	892
warezclient	1.00	1.00	1.00	890
warezmaster	1.00	1.00	1.00	20
accuracy			**1.00**	**125973**
macro avg	1.00	0.99	0.99	125973
weighted avg	1.00	1.00	1.00	125973

Table 18: decision tree on the multiclass test set (validation)

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	0
back	0.81	0.48	0.60	607
buffer_overflow	0.05	0.12	0.07	8
ftp_write	0.00	0.00	0.00	253
guess_passwd	0.03	1.00	0.06	37
httptunnel	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	2
ipsweep	0.99	0.97	0.98	145
land	0.71	0.62	0.67	8
loadmodule	0.00	0.00	0.00	17
mailbomb	0.00	0.00	0.00	0
mscan	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	10
named	0.00	0.00	0.00	0
neptune	1.00	0.93	0.96	4990
nmap	0.99	0.95	0.97	76
normal	0.94	0.73	0.82	12623
perl	0.50	0.25	0.33	4
phf	0.50	1.00	0.67	1
pod	0.93	0.72	0.81	53
portsweep	0.92	0.40	0.55	367
processtable	0.00	0.00	0.00	0
ps	0.00	0.00	0.00	0
rootkit	0.00	0.00	0.00	12
saint	0.00	0.00	0.00	0
satan	0.97	0.29	0.45	2457
sendmail	0.00	0.00	0.00	0
smurf	1.00	0.98	0.99	680
snmpgetattack	0.00	0.00	0.00	0
snmpguess	0.00	0.00	0.00	0
spy	0.00	0.00	0.00	3
sqlattack	0.00	0.00	0.00	0
teardrop	1.00	0.24	0.39	49
udpstorm	0.00	0.00	0.00	0
warezclient	0.00	0.00	0.00	140
warezmaster	0.00	0.50	0.00	2
worm	0.00	0.00	0.00	0
xlock	0.00	0.00	0.00	0
xsnoop	0.00	0.00	0.00	0
xterm	0.00	0.00	0.00	0
accuracy			**0.71**	**22544**
macro avg	0.28	0.25	0.23	22544
weighted avg	0.94	0.71	0.79	22544

Table 19: decision tree on the binary training set

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	58637
normal	1.00	1.00	1.00	67336
accuracy			**1.00**	**125973**
macro avg	1.00	1.00	1.00	125973
weigthed avg	1.00	1.00	1.00	125973

Table 20: decision tree on the binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.66	0.96	0.78	8879
normal	0.96	0.68	0.80	13665
accuracy			**0.79**	**22544**
macro avg	0.81	0.82	0.79	22544
weighted avg	0.84	0.79	0.79	22544

Table 21: decision tree on the 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	45932
Probe	1.00	1.00	1.00	11657
R2L	1.00	1.00	1.00	995
U2R	1.00	0.98	0.99	53
normal	1.00	1.00	1.00	67336
accuracy			**1.00**	**125973**
macro avg	1.00	1.00	1.00	125973
weighted avg	1.00	1.00	1.00	125973

Table 22: decision tree on the 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.80	0.96	0.87	6279
Probe	0.64	0.80	0.71	1929
R2L	0.08	0.98	0.14	222
U2R	0.25	0.55	0.35	31
normal	0.96	0.66	0.79	14083
accuracy			**0.76**	**22544**
macro avg	0.55	0.79	0.57	22544
weighted avg	0.88	0.76	0.80	22544

K-nearest neighbours:

Table 23: knn on the multiclass training set

	precision	recall	f1-score	support
back	0.95	0.96	0.96	949
buffer_overflow	0.00	0.00	0.00	0
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.92	0.70	0.80	70
imap	0.00	0.00	0.00	0
ipsweep	0.94	0.98	0.96	3451
land	0.00	0.00	0.00	0
loadmodule	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	0
neptune	1.00	0.98	0.99	41898
nmap	0.96	0.95	0.95	1514
normal	1.00	0.99	0.99	67620
perl	0.00	0.00	0.00	0
phf	0.00	0.00	0.00	0
pod	0.99	0.99	0.99	201
portsweep	0.85	0.99	0.92	2529
rootkit	0.00	0.00	0.00	0
satan	0.92	0.98	0.95	3400
smurf	1.00	0.99	0.99	2680
spy	0.00	0.00	0.00	0
teardrop	0.99	1.00	1.00	884
warezclient	0.84	0.96	0.90	777
warezmaster	0.00	0.00	0.00	0
accuracy			**0.99**	**125973**
macro avg	0.49	0.50	0.50	125973
weighted avg	0.99	0.99	0.99	125973

Table 24: knn on multiclass test set (validation)

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	0
back	0.89	0.40	0.55	804
buffer_overflow	0.00	0.00	0.00	0
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.30	0.99	0.46	368
httptunnel	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	0
ipsweep	0.96	0.79	0.87	173
land	0.00	0.00	0.00	0
loadmodule	0.00	0.00	0.00	0
mailbomb	0.00	0.00	0.00	0
mscan	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	0
named	0.00	0.00	0.00	0
neptune	1.00	0.86	0.93	5363
nmap	1.00	0.42	0.60	172
normal	0.96	0.69	0.80	13553
perl	0.00	0.00	0.00	0
phf	0.00	0.00	0.00	0
pod	0.88	0.69	0.77	52
portsweep	0.89	0.72	0.80	192
processtable	0.00	0.00	0.00	0
ps	0.00	0.00	0.00	0
rootkit	0.00	0.00	0.00	0
saint	0.00	0.00	0.00	0
satan	0.82	0.54	0.65	1105
sendmail	0.00	0.00	0.00	0
smurf	1.00	0.99	0.99	671
snmpgetattack	0.00	0.00	0.00	0
snmpguess	0.00	0.00	0.00	0
sqlattack	0.00	0.00	0.00	0
teardrop	0.67	0.18	0.28	45
udpstorm	0.00	0.00	0.00	0
warezclient	0.00	0.00	0.00	46
warezmaster	0.00	0.00	0.00	0
worm	0.00	0.00	0.00	0
xlock	0.00	0.00	0.00	0
xsnoop	0.00	0.00	0.00	0
xterm	0.00	0.00	0.00	0
accuracy			**0.72**	**22544**
macro avg	0.24	0.19	0.20	22544
weighted avg	0.94	0.72	0.81	22544

Table 25: knn on binary training set

	precision	recall	f1-score	support
abnormal	0.99	0.99	0.99	58450
normal	1.00	0.99	0.99	67523
accuracy			**0.99**	**125973**
macro avg	0.99	0.99	0.99	125973
weighted avg	0.99	0.99	0.99	125973

Table 26: knn on binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.66	0.92	0.77	9182
normal	0.93	0.67	0.78	13362
accuracy			**0.77**	**22544**
macro avg	0.79	0.80	0.77	22544
weighted avg	0.82	0.77	0.77	22544

Table 27: knn on 4-class training set

	precision	recall	f1-score	support
DoS	1.00	0.99	1.00	46282
Probe	0.96	0.99	0.98	11296
R2L	0.93	0.94	0.93	984
U2R	0.38	0.69	0.49	29
normal	1.00	1.00	1.00	67382
accuracy			**0.99**	**125973**
macro avg	0.85	0.92	0.88	125973
weighted avg	0.99	0.99	0.99	125973

Table 28: knn on 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.83	0.89	0.86	6900
Probe	0.53	0.66	0.59	1962
R2L	0.04	0.90	0.07	114
U2R	0.27	0.64	0.38	28
normal	0.93	0.66	0.77	13540
accuracy			**0.74**	**22544**
macro avg	0.52	0.75	0.53	22544
weighted avg	0.86	0.74	0.78	22544

Gaussian Naïve Bayes:

Table 29: Gaussian Naive Bayes on multiclass training set

	precision	recall	f1-score	support
back	1.00	0.08	0.15	12087
buffer_overflow	0.57	0.05	0.09	331
ftp_write	1.00	0.04	0.07	208
guess_passwd	1.00	0.95	0.97	56
imap	1.00	0.85	0.92	13
ipsweep	0.99	0.33	0.49	10793
land	1.00	0.72	0.84	25
loadmodule	0.78	0.03	0.05	265
multihop	0.43	0.04	0.07	83
neptune	1.00	1.00	1.00	41128
nmap	0.18	0.51	0.27	533
normal	0.65	0.98	0.78	44889
perl	1.00	1.00	1.00	3
phf	1.00	1.00	1.00	4
pod	1.00	0.91	0.95	222
portsweep	0.87	0.59	0.71	4312
rootkit	0.60	0.01	0.01	1109
satan	0.01	0.72	0.02	53
smurf	1.00	0.92	0.96	2861
spy	1.00	1.00	1.00	2
teardrop	1.00	0.29	0.45	3055
warezclient	0.35	0.12	0.18	2580
warezmaster	1.00	0.01	0.03	1361
accuracy			**0.77**	**125973**
macro avg	0.80	0.53	0.52	125973
weighted avg	0.85	0.77	0.73	125973

Table 30: Gaussian Naive Bayes on multiclass test set (validation)

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	0
back	0.38	0.06	0.10	2358
buffer_overflow	0.00	0.00	0.00	0
ftp_write	0.33	0.01	0.02	84
guess_passwd	0.02	1.00	0.04	27
httptunnel	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	0
ipsweep	0.99	0.22	0.35	646
land	1.00	1.00	1.00	7
loadmodule	0.00	0.00	0.00	98
mailbomb	0.00	0.00	0.00	0
mscan	0.00	0.00	0.00	0
multihop	0.22	0.18	0.20	22
named	0.00	0.00	0.00	0
neptune	0.98	0.97	0.98	4701
nmap	1.00	0.41	0.58	179
normal	0.62	0.59	0.60	10215
perl	0.00	0.00	0.00	0
phf	0.50	1.00	0.67	1
pod	0.98	0.65	0.78	62
portsweep	0.74	0.11	0.19	1070
processtable	0.00	0.00	0.00	0
ps	0.00	0.00	0.00	0
rootkit	0.15	0.00	0.01	489
saint	0.00	0.00	0.00	0
satan	0.00	0.14	0.01	21
sendmail	0.00	0.00	0.00	0
smurf	0.98	0.97	0.97	669
snmpgetattack	0.00	0.00	0.00	0
snmpguess	0.00	0.00	0.00	0
sqlattack	0.00	0.00	0.00	0
teardrop	1.00	0.01	0.02	1335
udpstorm	0.00	0.00	0.00	0
warezclient	0.00	0.00	0.00	216
warezmaster	0.24	0.66	0.35	344
worm	0.00	0.00	0.00	0
xlock	0.00	0.00	0.00	0
xsnoop	0.00	0.00	0.00	0
xterm	0.00	0.00	0.00	0
accuracy			**0.53**	**22544**
macro avg	0.26	0.20	0.18	22544
weighted avg	0.70	0.53	0.55	22544

Table 31: Gaussian Naive Bayes on binary training set

	precision	recall	f1-score	support
abnormal	0.66	1.00	0.79	38781
normal	1.00	0.77	0.87	87192
accuracy			**0.84**	**125973**
macro avg	0.83	0.88	0.83	125973
weighted avg	0.89	0.84	0.85	125973

Table 32: Gaussian Naive Bayes on binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.22	0.98	0.36	2909
normal	0.99	0.49	0.66	19635
accuracy			**0.55**	**22544**
macro avg	0.61	0.74	0.51	22544
weigthed avg	0.89	0.55	0.62	22544

Table 33: Gaussian Naive Bayes on 4-class training set

	precision	recall	f1-score	support
DoS	0.89	0.99	0.93	41369
Probe	0.13	0.96	0.23	1594
R2L	0.51	0.02	0.04	25292
U2R	1.00	0.01	0.02	5712
normal	0.58	0.75	0.65	52006
accuracy			**0.65**	**125973**
macro avg	0.62	0.54	0.38	125973
weighted avg	0.68	0.65	0.59	125973

Table 34: Gaussian Naive Bayes on 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.39	0.88	0.54	3346
Probe	0.08	0.91	0.15	215
R2L	0.32	0.14	0.20	6499
U2R	0.67	0.04	0.08	1074
normal	0.55	0.47	0.50	11410
accuracy			**0.42**	**22544**
macro avg	0.40	0.49	0.30	22544
weighted avg	0.46	0.42	0.40	22544

Multi-layer perceptron:

Table 35: MLP on multiclass training set

	precision	recall	f1-score	support
back	0.99	0.98	0.99	963
buffer_overflow	0.77	1.00	0.87	23
ftp_write	0.75	1.00	0.86	6
guess_passwd	1.00	1.00	1.00	53
imap	1.00	0.92	0.96	12
ipsweep	0.99	0.99	0.99	3622
land	1.00	0.72	0.84	25
loadmodule	0.78	1.00	0.88	7
multihop	0.57	0.80	0.67	5
neptune	1.00	1.00	1.00	41215
nmap	0.96	0.99	0.98	1455
normal	1.00	1.00	1.00	67388
perl	1.00	1.00	1.00	3
phf	1.00	1.00	1.00	4
pod	0.99	1.00	0.99	198
portsweep	1.00	1.00	1.00	2927
rootkit	0.40	1.00	0.57	4
satan	0.99	1.00	0.99	3613
smurf	1.00	1.00	1.00	2640
spy	1.00	1.00	1.00	2
teardrop	1.00	1.00	1.00	892
warezclient	0.96	0.96	0.96	892
warezmaster	1.00	0.83	0.91	24
accuracy			**1.00**	**125973**
macro avg	0.92	0.96	0.93	125973
weighted avg	1.00	1.00	1.00	125973

Table 36: MLP on multiclass test set (validation)

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	0
back	0.94	0.41	0.57	822
buffer_overflow	0.05	0.50	0.09	2
ftp_write	0.33	0.01	0.01	131
guess_passwd	0.00	0.25	0.00	4
httptunnel	0.00	0.00	0.00	0
imap	0.00	0.00	0.00	32
ipsweep	0.97	0.83	0.90	165
land	1.00	0.39	0.56	18
loadmodule	0.00	0.00	0.00	9
mailbomb	0.00	0.00	0.00	0
mscan	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	10
named	0.00	0.00	0.00	0
neptune	1.00	0.95	0.98	4863
nmap	0.99	0.43	0.60	168
normal	0.97	0.68	0.80	13994
perl	0.50	0.33	0.40	3
phf	0.50	0.25	0.33	4
pod	0.88	0.71	0.78	51
portsweep	0.94	0.35	0.51	421
processtable	0.00	0.00	0.00	0
ps	0.00	0.00	0.00	0
rootkit	0.00	0.00	0.00	3
saint	0.00	0.00	0.00	0
satan	0.77	0.56	0.65	1025
sendmail	0.00	0.00	0.00	0
smurf	1.00	0.99	0.99	674
snmpgetattack	0.00	0.00	0.00	0
snmpguess	0.00	0.00	0.00	0
spy	0.00	0.00	0.00	32
sqlattack	0.00	0.00	0.00	0
teardrop	1.00	0.24	0.39	49
udpstorm	0.00	0.00	0.00	0
warezclient	0.00	0.00	0.00	10
warezmaster	0.04	0.72	0.08	54
worm	0.00	0.00	0.00	0
xlock	0.00	0.00	0.00	0
xsnoop	0.00	0.00	0.00	0
xterm	0.00	0.00	0.00	0
accuracy			**0.72**	**22544**
macro avg	0.30	0.21	0.22	22544
weighted avg	0.96	0.72	0.81	22544

Table 37: MLP on binary training set

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	58571
normal	1.00	1.00	1.00	67402
accuracy			**1.00**	**125973**
macro avg	1.00	1.00	1.00	125973
weighted avg	1.00	1.00	1.00	125973

Table 38: MLP on binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.66	0.97	0.78	8769
normal	0.97	0.68	0.80	13775
accuracy			**0.79**	**22544**
macro avg	0.81	0.82	0.79	22544
weighted avg	0.85	0.79	0.79	22544

Table 39: MLP on 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	45902
Probe	1.00	1.00	1.00	11660
R2L	0.94	0.97	0.96	956
U2R	0.75	0.95	0.84	41
normal	1.00	1.00	1.00	67414
accuracy			**1.00**	**125973**
macro	0.94	0.98	0.96	125973
weighted avg	1.00	1.00	1.00	125973

Table 40: MLP on 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.82	0.97	0.89	6365
Probe	0.59	0.84	0.69	1686
R2L	0.11	0.79	0.19	391
U2R	0.31	0.88	0.46	24
normal	0.97	0.67	0.79	14078
accuracy			**0.77**	**22544**
macro avg	0.56	0.83	0.61	22544
weighted avg	0.89	0.77	0.80	22544

C.2. Case B: splitting the *KDDTrain+* for training and test sets

Logistic Regression:

Table 41: logistic regression on the split multiclass training set

	precision	recall	f1-score	support
back	0.99	0.95	0.97	790
buffer_overflow	0.70	0.82	0.76	17
ftp_write	0.17	0.50	0.25	2
guess_passwd	0.97	0.93	0.95	42
imap	0.38	1.00	0.55	3
ipsweep	0.97	0.97	0.97	2924
land	1.00	0.70	0.82	20
loadmodule	0.11	0.25	0.15	4
multihop	0.00	0.00	0.00	1
neptune	1.00	1.00	1.00	32966
nmap	0.95	0.92	0.94	1246
normal	0.99	0.99	0.99	53988
perl	0.00	0.00	0.00	0
phf	1.00	0.57	0.73	7
pod	0.99	1.00	0.99	158
portsweep	0.98	1.00	0.99	2280
rootkit	0.17	0.33	0.22	3
satan	0.94	0.98	0.96	2780
smurf	1.00	0.99	0.99	2164
spy	0.00	0.00	0.00	0
teardrop	1.00	1.00	1.00	731
warezclient	0.80	0.88	0.84	640
warezmaster	0.58	0.58	0.58	12
accuracy			**0.99**	**100778**
macro	0.68	0.71	0.68	100778
weighted	0.99	0.99	0.99	100778

Table 42: logistic regression on the split multiclass test set (validation)

	precision	recall	f1-score	support
back	0.97	0.94	0.95	203
buffer_overflow	0.50	0.83	0.62	6
ftp_write	0.50	0.50	0.50	2
guess_passwd	0.77	1.00	0.87	10
imap	0.33	1.00	0.50	1
ipsweep	0.97	0.96	0.96	710
land	1.00	0.80	0.89	5
loadmodule	0.00	0.00	0.00	1
multihop	0.00	0.00	0.00	0
neptune	1.00	1.00	1.00	8266
nmap	0.98	0.92	0.95	309
normal	0.99	0.99	0.99	13485
perl	1.00	1.00	1.00	1
phf	0.00	0.00	0.00	1
pod	0.98	1.00	0.99	40
portsweep	0.98	1.00	0.99	613
rootkit	0.00	0.00	0.00	1
satan	0.93	0.99	0.96	681
smurf	1.00	0.98	0.99	517
spy	0.00	0.00	0.00	0
teardrop	1.00	1.00	1.00	161
warezclient	0.82	0.87	0.84	174
warezmaster	1.00	1.00	1.00	8
accuracy			**0.99**	**25195**
macro	0.68	0.73	0.70	25195
weighted	0.99	0.99	0.99	25195

Table 43: logistic regression on the split binary training set

	precision	recall	f1-score	support
abnormal	0.96	0.98	0.97	46228
normal	0.98	0.97	0.98	54550
accuracy			**0.97**	**100778**
macro avg	0.97	0.97	0.97	100778
weighted avg	0.97	0.97	0.97	100778

Table 44: logistic regression on the split binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.97	0.98	0.97	11575
normal	0.98	0.97	0.98	13620
accuracy			**0.97**	**25195**
macro avg	0.97	0.97	0.97	25195
weighted avg	0.97	0.97	0.97	25195

Table 45: logistic regression on the split 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	36845
Probe	0.96	0.98	0.97	9161
R2L	0.74	0.81	0.77	718
U2R	0.54	0.83	0.66	24
normal	0.99	0.99	0.99	54030
accuracy			**0.99**	**100778**
macro avg	0.85	0.92	0.88	100778
weighted avg	0.99	0.99	0.99	100778

Table 46: logistic regression on the split 4-class test set (validation)

	precision	recall	f1-score	support
DoS	1.00	0.99	1.00	9191
Probe	0.96	0.98	0.97	2293
R2L	0.76	0.83	0.79	198
U2R	0.47	0.78	0.58	9
normal	0.99	0.99	0.99	13504
accuracy			**0.99**	**25195**
macro avg	0.84	0.91	0.87	25195
weighted avg	0.99	0.99	0.99	25195

Decision Tree:

Table 47: decision tree on the split multiclass training set

	precision	recall	f1-score	support
back	1.00	1.00	1.00	760
buffer_overflow	1.00	1.00	1.00	20
ftp_write	1.00	1.00	1.00	6
guess_passwd	1.00	1.00	1.00	40
imap	1.00	1.00	1.00	8
ipsweep	1.00	1.00	1.00	2901
land	1.00	0.82	0.90	17
loadmodule	1.00	1.00	1.00	9
multihop	1.00	1.00	1.00	5
neptune	1.00	1.00	1.00	32955
nmap	1.00	1.00	1.00	1196
normal	1.00	1.00	1.00	53885
perl	1.00	1.00	1.00	2
phf	1.00	1.00	1.00	4
pod	0.99	1.00	1.00	159
portsweep	1.00	1.00	1.00	2307
rootkit	0.83	1.00	0.91	5
satan	1.00	1.00	1.00	2910
smurf	1.00	1.00	1.00	2141
spy	1.00	1.00	1.00	1
teardrop	1.00	1.00	1.00	731
warezclient	1.00	1.00	1.00	704
warezmaster	1.00	1.00	1.00	12
accuracy			**1.00**	**100778**
macro	0.99	0.99	0.99	100778
weighted	1.00	1.00	1.00	100778

Table 48: decision tree on the split multiclass test set (validation)

	precision	recall	f1-score	support
back	1.00	0.99	1.00	197
buffer_overflow	0.80	0.73	0.76	11
ftp_write	0.00	0.00	0.00	4
guess_passwd	0.77	1.00	0.87	10
imap	1.00	1.00	1.00	3
ipsweep	0.99	0.99	0.99	704
land	0.75	0.60	0.67	5
loadmodule	0.00	0.00	0.00	4
multihop	0.00	0.00	0.00	0
neptune	1.00	1.00	1.00	8261
nmap	0.98	0.99	0.98	290
normal	1.00	1.00	1.00	13457
perl	0.00	0.00	0.00	0
phf	0.00	0.00	0.00	2
pod	1.00	1.00	1.00	41
portsweep	0.99	0.99	0.99	627
rootkit	0.00	0.00	0.00	1
satan	0.99	0.99	0.99	720
smurf	1.00	1.00	1.00	505
spy	0.00	0.00	0.00	1
teardrop	1.00	1.00	1.00	161
warezclient	0.97	0.99	0.98	182
warezmaster	1.00	0.89	0.94	9
accuracy			**1.00**	**25195**
macro avg	0.66	0.66	0.66	25195
weighted avg	1.00	1.00	1.00	25195

Table 49: decision tree on the split binary training set

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	46897
normal	1.00	1.00	1.00	53881
accuracy			**1.00**	**100778**
macro avg	1.00	1.00	1.00	100778
weigthed avg	1.00	1.00	1.00	100778

Table 50: decision tree on the split binary test set (validation)

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	11736
normal	1.00	1.00	1.00	13459
accuracy			**1.00**	**25195**
macro avg	1.00	1.00	1.00	25195
weighted avg	1.00	1.00	1.00	25195

Table 51: decision tree on the split 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	36765
Probe	1.00	1.00	1.00	9314
R2L	1.00	1.00	1.00	780
U2R	1.00	0.97	0.99	38
normal	1.00	1.00	1.00	53881
accuracy			**1.00**	**100778**
macro avg	1.00	0.99	1.00	100778
weighted avg	1.00	1.00	1.00	100778

Table 52: decision tree on the split 4-class test set (validation)

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	9164
Probe	0.99	1.00	0.99	2339
R2L	0.96	0.97	0.97	213
U2R	0.67	0.62	0.65	16
normal	1.00	1.00	1.00	13463
accuracy			**1.00**	**25195**
macro avg	0.92	0.92	0.92	25195
weighted avg	1.00	1.00	1.00	25195

K-nearest neighbours:

Table 53: knn on the split multiclass training set

	precision	recall	f1-score	support
back	0.96	0.96	0.96	760
buffer_overflow	0.00	0.00	0.00	0
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.97	0.66	0.79	59
imap	0.00	0.00	0.00	0
ipsweep	0.93	0.99	0.96	2728
land	0.00	0.00	0.00	0
loadmodule	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	0
neptune	1.00	0.98	0.99	33544
nmap	0.96	0.95	0.95	1217
normal	1.00	0.99	0.99	54113
perl	0.00	0.00	0.00	0
phf	0.00	0.00	0.00	0
pod	0.99	0.98	0.98	161
portsweep	0.84	0.99	0.91	1961
rootkit	0.00	0.00	0.00	0
satan	0.92	0.98	0.95	2726
smurf	1.00	0.99	0.99	2171
spy	0.00	0.00	0.00	0
teardrop	0.99	1.00	1.00	725
warezclient	0.84	0.96	0.90	613
warezmaster	0.00	0.00	0.00	0
accuracy			**0.99**	**100778**
macro avg	0.50	0.50	0.49	100778
weighted avg	0.99	0.99	0.99	100778

Table 54: knn on the split multiclass test set (validation)

	precision	recall	f1-score	support
back	0.94	0.98	0.96	189
buffer_overflow	0.00	0.00	0.00	0
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.85	0.55	0.67	20
imap	0.00	0.00	0.00	0
ipsweep	0.93	0.99	0.96	665
land	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	0
neptune	1.00	0.98	0.99	8411
nmap	0.97	0.94	0.96	302
normal	1.00	0.99	0.99	13531
perl	0.00	0.00	0.00	0
pod	0.98	1.00	0.99	40
portsweep	0.84	1.00	0.91	524
rootkit	0.00	0.00	0.00	0
satan	0.91	0.98	0.94	672
smurf	1.00	0.97	0.98	522
spy	0.00	0.00	0.00	0
teardrop	0.99	1.00	0.99	159
warezclient	0.83	0.96	0.89	160
warezmaster	0.00	0.00	0.00	0
accuracy			**0.99**	**25195**
macro avg	0.53	0.54	0.53	25195
weighted avg	0.99	0.99	0.99	25195

Table 55: knn on the split binary training set

	precision	recall	f1-score	support
abnormal	0.99	0.99	0.99	46759
normal	0.99	0.99	0.99	54019
accuracy			**0.99**	**100778**
macro avg	0.99	0.99	0.99	100778
weighted avg	0.99	0.99	0.99	100778

Table 56: knn on the split binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.99	0.99	0.99	11691
normal	0.99	0.99	0.99	13504
accuracy			**0.99**	**25195**
macro avg	0.99	0.99	0.99	25195
weighted avg	0.99	0.99	0.99	25195

Table 57: knn on the split 4-class training set

	precision	recall	f1-score	support
DoS	1.00	0.98	0.99	37360
Probe	0.92	0.99	0.96	8647
R2L	0.81	0.94	0.87	676
U2R	0.00	0.00	0.00	0
normal	1.00	0.99	0.99	54095
accuracy			**0.99**	**100778**
macro avg	0.75	0.78	0.76	100778
weighted avg	0.99	0.99	0.99	100778

Table 58: knn on the split 4-class test set (validation)

	precision	recall	f1-score	support
DoS	1.00	0.98	0.99	9321
Probe	0.92	0.99	0.95	2165
R2L	0.79	0.92	0.85	183
U2R	0.00	0.00	0.00	0
normal	1.00	0.99	0.99	13526
accuracy			**0.99**	**25195**
macro avg	0.74	0.78	0.76	25195
weighted avg	0.99	0.99	0.99	25195

Gaussian Naïve Bayes:

Table 59: Gaussian Naive Bayes on the split multiclass training set

	precision	recall	f1-score	support
back	1.00	0.08	0.15	9634
buffer_overflow	0.35	0.09	0.14	82
ftp_write	1.00	0.04	0.08	147
guess_passwd	1.00	1.00	1.00	40
imap	1.00	0.80	0.89	10
ipsweep	0.99	0.31	0.47	9198
land	1.00	0.70	0.82	20
loadmodule	0.89	0.04	0.08	197
multihop	0.40	0.04	0.07	54
neptune	1.00	1.00	1.00	32873
nmap	0.18	0.43	0.26	510
normal	0.64	0.98	0.78	35421
perl	1.00	1.00	1.00	2
phf	1.00	1.00	1.00	4
pod	1.00	0.90	0.95	177
portsweep	0.89	0.58	0.70	3529
rootkit	0.83	0.00	0.01	1108
satan	0.01	0.66	0.02	53
smurf	1.00	0.93	0.96	2307
spy	1.00	1.00	1.00	1
teardrop	1.00	0.30	0.46	2448
warezclient	0.37	0.13	0.20	1924
warezmaster	1.00	0.01	0.02	1039
accuracy			**0.76**	**100778**
macro avg	0.81	0.52	0.52	100778
weighted avg	0.85	0.76	0.72	100778

Table 60: Gaussian Naive Bayes on the split multiclass test set (validation)

	precision	recall	f1-score	support
back	1.00	0.08	0.15	9634
buffer_overflow	0.35	0.09	0.14	82
ftp_write	1.00	0.04	0.08	147
guess_passwd	1.00	1.00	1.00	40
imap	1.00	0.80	0.89	10
ipsweep	0.99	0.31	0.47	9198
land	1.00	0.70	0.82	20
loadmodule	0.89	0.04	0.08	197
multihop	0.40	0.04	0.07	54
neptune	1.00	1.00	1.00	32873
nmap	0.18	0.43	0.26	510
normal	0.64	0.98	0.78	35421
perl	1.00	1.00	1.00	2
phf	1.00	1.00	1.00	4
pod	1.00	0.90	0.95	177
portsweep	0.89	0.58	0.70	3529
rootkit	0.83	0.00	0.01	1108
satan	0.01	0.66	0.02	53
smurf	1.00	0.93	0.96	2307
spy	1.00	1.00	1.00	1
teardrop	1.00	0.30	0.46	2448
warezclient	0.37	0.13	0.20	1924
warezmaster	1.00	0.01	0.02	1039
accuracy			**0.76**	**100778**
macro avg	0.81	0.52	0.52	100778
weighted avg	0.85	0.76	0.72	100778

Table 61: Gaussian Naive Bayes on the split binary training set

	precision	recall	f1-score	support
abnormal	0.67	1.00	0.80	31514
normal	1.00	0.78	0.87	69264
accuracy			**0.85**	**100778**
macro avg	0.83	0.89	0.84	100778
weighted avg	0.90	0.85	0.85	100778

Table 62: Gaussian Naive Bayes on the split binary test set (validation)

	precision	recall	f1-score	support
abnormal	0.67	1.00	0.80	7876
normal	1.00	0.78	0.87	17319
accuracy			**0.85**	**25195**
macro avg	0.83	0.89	0.84	25195
weigthed avg	0.90	0.85	0.85	25195

Table 63: Gaussian Naive Bayes on the split 4-class training set

	precision	recall	f1-score	support
DoS	0.89	0.99	0.94	33310
Probe	0.18	0.97	0.30	1718
R2L	0.55	0.02	0.04	21711
U2R	1.00	0.01	0.02	4102
normal	0.56	0.75	0.64	39937
accuracy			**0.65**	**100778**
macro avg	0.64	0.55	0.39	100778
weighted avg	0.68	0.65	0.58	100778

Table 64: Gaussian Naive Bayes on the split 4-class test set (validation)

	precision	recall	f1-score	support
DoS	0.89	0.99	0.94	8266
Probe	0.19	0.97	0.32	456
R2L	0.57	0.02	0.04	5441
U2R	0.87	0.01	0.03	1017
normal	0.56	0.75	0.64	10015
accuracy			**0.65**	**25195**
macro avg	0.61	0.55	0.39	25195
weighted avg	0.68	0.65	0.58	25195

Multi-layer perceptron:

Table 65: MLP on the split multiclass training set

	precision	recall	f1-score	support
back	0.99	1.00	1.00	755
buffer_overflow	0.90	0.95	0.92	19
ftp_write	0.67	1.00	0.80	4
guess_passwd	1.00	1.00	1.00	40
imap	1.00	1.00	1.00	8
ipsweep	1.00	0.98	0.99	2935
land	1.00	0.70	0.82	20
loadmodule	0.78	1.00	0.88	7
multihop	0.40	1.00	0.57	2
neptune	1.00	1.00	1.00	32956
nmap	0.95	0.99	0.97	1152
normal	1.00	1.00	1.00	53990
perl	1.00	1.00	1.00	2
phf	1.00	1.00	1.00	4
pod	0.99	1.00	0.99	158
portsweep	1.00	1.00	1.00	2305
rootkit	0.67	1.00	0.80	4
satan	0.99	1.00	1.00	2886
smurf	1.00	1.00	1.00	2133
spy	1.00	1.00	1.00	1
teardrop	1.00	1.00	1.00	731
warezclient	0.88	0.95	0.91	650
warezmaster	1.00	0.75	0.86	16
accuracy			**1.00**	**100778**
macro avg	0.92	0.97	0.93	100778
weighted avg	1.00	1.00	1.00	100778

Table 66: MLP on the split multiclass test set (validation)

	precision	recall	f1-score	support
back	0.98	0.99	0.99	194
buffer_overflow	0.50	0.62	0.56	8
ftp_write	0.50	0.50	0.50	2
guess_passwd	0.85	1.00	0.92	11
imap	1.00	1.00	1.00	3
ipsweep	0.99	0.97	0.98	718
land	1.00	0.80	0.89	5
multihop	0.00	0.00	0.00	0
neptune	1.00	1.00	1.00	8259
nmap	0.94	0.99	0.96	279
normal	1.00	0.99	1.00	13499
perl	1.00	1.00	1.00	1
pod	0.98	1.00	0.99	40
portsweep	0.99	0.99	0.99	622
rootkit	0.00	0.00	0.00	0
satan	0.98	0.99	0.98	713
smurf	0.99	0.99	0.99	504
spy	0.00	0.00	0.00	0
teardrop	1.00	1.00	1.00	161
warezclient	0.86	0.96	0.91	166
warezmaster	1.00	0.80	0.89	10
accuracy			**1.00**	**25195**
macro avg	0.79	0.79	0.79	25195
weighted avg	1.00	1.00	1.00	25195

Table 67: MLP on the split binary training set

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	46822
normal	1.00	1.00	1.00	53956
accuracy			**1.00**	**100778**
macro avg	1.00	1.00	1.00	100778
weighted avg	1.00	1.00	1.00	100778

Table 68: MLP on the split binary test set (validation)

	precision	recall	f1-score	support
abnormal	1.00	1.00	1.00	11703
normal	1.00	1.00	1.00	13492
accuracy			**1.00**	**25195**
macro avg	1.00	1.00	1.00	25195
weighted avg	1.00	1.00	1.00	25195

Table 69: MLP on the split 4-class training set

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	36759
Probe	1.00	1.00	1.00	9308
R2L	0.96	0.94	0.95	793
U2R	0.86	0.91	0.89	35
normal	1.00	1.00	1.00	53883
accuracy			**1.00**	**100778**
macro	0.96	0.97	0.97	100778
weighted avg	1.00	1.00	1.00	100778

Table 70: MLP on the split 4-class test set (validation)

	precision	recall	f1-score	support
DoS	1.00	1.00	1.00	9165
Probe	0.99	0.99	0.99	2338
R2L	0.94	0.96	0.95	211
U2R	0.40	1.00	0.57	6
normal	1.00	1.00	1.00	13475
accuracy			**1.00**	**25195**
macro avg	0.87	0.99	0.90	25195
weighted avg	1.00	1.00	1.00	25195

References

[1] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *Network Traffic Anomaly Detection and Prevention: concepts, techniques and tools*. 2017. [Online]. Available: http://www.springer.com/series/4198

[2] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein, "Machine learning in network anomaly detection: A survey," *IEEE Access*, vol. 9, pp. 152379–152396, 2021, doi: 10.1109/ACCESS.2021.3126834.

[3] A. Drewek-Ossowicka, M. Pietrołaj, and J. Rumiński, "A survey of neural networks usage for intrusion detection systems," *J Ambient Intell Humaniz Comput*, vol. 12, no. 1, pp. 497–514, Jan. 2021, doi: 10.1007/s12652-020-02014-x.

[4] R. Bala and R. Nagpal, "A review on KDDCUP99 and NSL-KDD dataset," 2019, doi: 10.26483/ijarcs.v10i2.6395.

[5] "What is Supervised Learning? | IBM." https://www.ibm.com/cloud/learn/supervised-learning

[6] "Supervised Machine Learning: What is, Algorithms with Examples." https://www.guru99.com/supervised-machine-learning.html

[7] A. Binbusayyis and T. Vaiyapuri, "Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class SVM," *Applied Intelligence*, vol. 51, no. 10, pp. 7094–7108, Oct. 2021, doi: 10.1007/s10489-021-02205-9.

[8] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with LSTM neural networks," *IEEE Trans Neural Netw Learn Syst*, vol. 31, no. 8, pp. 3127–3141, Aug. 2020, doi: 10.1109/TNNLS.2019.2935975.

[9] S. Rawat, A. Srinivasan, V. Ravi, and U. Ghosh, "Intrusion detection systems using classical machine learning techniques vs integrated unsupervised feature learning and deep neural network," *Internet Technology Letters*, vol. 5, no. 1, Jan. 2022, doi: 10.1002/itl2.232.

[10] R. Abdulhammed, M. Faezipour, A. Abuzneid, and A. Abumallouh, "Deep and Machine Learning Approaches for Anomaly-Based Intrusion Detection of Imbalanced Network Traffic," *IEEE Sens Lett*, vol. 3, no. 1, Jan. 2019, doi: 10.1109/LSENS.2018.2879990.

[11] J. J. Estévez-Pereira, D. Fernández, and F. J. Novoa, "Network Anomaly Detection Using Machine Learning Techniques," Aug. 2020, p. 8. doi: 10.3390/proceedings2020054008.

[12] S. Gurung, M. K. Ghose, and A. Subedi, "Deep Learning Approach on Network Intrusion Detection System using NSL-KDD Dataset," *Computer Network and Information Security*, vol. 3, pp. 8–14, 2019, doi: 10.5815/ijcnis.2019.03.02.

[13] O. Jamal Ibrahim *et al.*, “Network intrusion detection: a comparative study of four classifiers using the NSL-KDD and KDD’99 datasets,” *J. Phys*, p. 12043, 2022, doi: 10.1088/1742-6596/2161/1/012043.

[14] A. Ng, “CS229 Lecture Notes - Supervised Machine Learning,” in *Stanford Machine Learning Course*, 2019.

[15] “Advantages and Disadvantages of Logistic Regression.” https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/

[16] “Machine Learning Decision Tree Classification Algorithm - Javatpoint.” https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

[17] “K-Nearest Neighbours - GeeksforGeeks.” https://www.geeksforgeeks.org/k-nearest-neighbours/

[18] “What Is K-Nearest Neighbor? An ML Algorithm to Classify Data.” https://learn.g2.com/k-nearest-neighbor

[19] “Naive Bayes classifier - Wikipedia.” https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[20] “Gaussian Naive Bayes: What You Need to Know? | upGrad blog.” https://www.upgrad.com/blog/gaussian-naive-bayes/#scroll-top

[21] “Multilayer Perceptron Definition | DeepAI.” https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

[22] “Multilayer perceptron - Wikipedia.” https://en.wikipedia.org/wiki/Multilayer_perceptron

[23] “Why MultiLayer Perceptron/Neural Network?,” in *MIT Media Lab MAS Project notes*,

[24] “Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning | by Uniqtech | Data Science Bootcamp | Medium.” https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1

[25] “A Deeper Dive into the NSL-KDD Data Set | by Gerry Saporito | Towards Data Science.” https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657

[26] Τμήμα Φυσικής ΑΠΘ, “Δίκτυα Επικοινωνίας και Υπολογιστών,”

[27] “Creating dummy variables in Python - AskPython.” https://www.askpython.com/python/examples/creating-dummy-variables

[28] “pandas.DataFrame.corr — pandas 1.4.3 documentation.” https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

[29] “sklearn.preprocessing.StandardScaler — scikit-learn 1.1.1 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[30] “sklearn.model_selection.train_test_split — scikit-learn 1.1.1 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[31] “Understanding a Classification Report For Your Machine Learning Model | by Shivam Kohli | Medium.” https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397

[32] “Compute Classification Report and Confusion Matrix in Python - GeeksforGeeks.” https://www.geeksforgeeks.org/compute-classification-report-and-confusion-matrix-in-python/

[33] “sklearn.metrics.precision_recall_fscore_support — scikit-learn 1.1.1 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support

[34] “3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 1.1.1 documentation.” https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

[35] S. Naseer *et al.*, “Enhanced network anomaly detection based on deep neural networks,” *IEEE Access*, vol. 6, pp. 48231–48246, Aug. 2018, doi: 10.1109/ACCESS.2018.2863036.

[36] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, “BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset,” *IEEE Access*, vol. 8, pp. 29575–29585, 2020, doi: 10.1109/ACCESS.2020.2972627.