INSE 6180 Projet

Intrusion detection on NSL-KDD

Course Name
INSE 6180 - Security and Privacy Implications of Data Mining

Instructor Name
Professor Nizar Bouguila

Student Names

Maryam Al Shami (40225059)
Mohamad Al Adraa (40201457)

"We certify that this submission is our original work and meets the Faculty's Expectations of Originality"

Submission date: 15/04/2022

Contents

Chapter 1

Abstract

With the advent of cloud computing, social networks and IoT there is a tremendous amount of network traffic generated everyday. So, there is a high possibility that some of this traffic may be suspicious, as the number of cyberattacks has increased steadily over the past few years. Thus, monitoring the traffic dynamically is extremely necessary to detect attacks ahead of time. By definition, an Intrusion Detection System (IDS) is designed to recognize suspicious activities and report them back to the administrators, so they can take the appropriate actions. Recently, IDS based on machine learning have been used, but ML-based IDS are hindered by the need for datasets. Additionally, the detection process needs to be carried out very quickly, since learning about the attack at a later stage will not be helpful. To carry out this study, we will use NSL-KDD dataset. We believe that the data preprocessing stage is a fundamental and crucial step in getting high accuracy models before the model selection process can begin. In this work, we will apply the process of feature selection to select the most influential features of the NSL-KDD, which may enhance the model's accuracy and speed up the training process. A variety of popular data mining algorithms will be used to perform intrusion detection in our study, including k-nearest neighbor and Naive Bayes. These techniques will be implemented starting from scratch, their pros and cons will be discussed, and their performance on the chosen dataset will be analyzed.

Chapter 2

Introduction

Due to the rapid growth of internet construction, identifying intrusions that threaten network security is critical. By analysing patterns of captured data, the Intrusion Detection System (IDS) was proposed in 1980 to provide solid protection for equipment against malicious software attacks such as denial of service (DoS). IDS can detect attacks and, for example, denies or prevents illegitimate traffic when it performs denial of service (DoS). In general, intrusion detection can be thought of as the process of resolving a classification problem.

2.1 Contribution

The main contributions of our project are summarized as following:

1. The complete NSL-KDD dataset is used rather than just a portion of it, as many previous studies did.
2. We proposed and applied a feature selection process with the use of the SelectFromModel technique.
3. We split our classification problem into BINARY & MULTI-CLASS, and we compare both approaches.
4. The data mining algorithms applied are implemented from scratch using python language.

2.2 Problem

The evolution of malicious software (malware) poses a critical challenge to the design of intrusion detection systems (IDS). Malicious attacks have become more sophisticated and the foremost challenge is to identify unknown and obfuscated malware, as the malware authors use different evasion techniques for information concealing to prevent detection by an IDS. However, existing traditional detection mechanisms suffer from high false negative predictions rate as well as slow detection. Hence, it is vital to find a novel intrusion detection system improving the detection mechanism and accelerating the detection process.

2.3 Proposed Solution

We propose an Intrusion Detection System that is Machine Learning-based and employs binary and multi-class classification models. Our goal in this project is to develop a machine learning model that reduces the number of false negative predictions. Similarly, this model will allow the system to detect attacks rapidly, so that the administrator can take the necessary action before the attack is complete.

2.4 Motivations

It is predicted that by 2023, there will be around 2.3 billion devices connected to the internet, generating petabytes of data. Detecting malicious traffic through an IDS is one of the most effective techniques, and this motivates us to explore the different types of IDS and in particular the ML-based ones. Additionally, the NSL-KDD dataset's hybrid mode - combining numerical and categorical information - motivates us to explore novel feature selection and data preprocessing techniques. Moreover, demonstrating the potential utility of these techniques in detecting attacks is highly intriguing.

The rest of this report is structured as follows. In chapter 3, we discuss the relevant studies in the realm of intrusion detection. In chapter 4, we describe the dataset as well as the attacks covered in our analysis. Moreover, we discuss the methodology we followed to solve the stated data mining problem, in chapter 5. Lastly, we discuss the performance assessment metrics, and compare binary & multi-class approaches, and then we conclude.

Chapter 3

Related Work

Nkiama et al. [8] proposed a technique for selecting and identifying relevant features on the NSL-KDD dataset, based on the score each feature established during the selection process, to exclude non-relevant features and identify those that would contribute to the improvement of the detection rate. They employed a recursive feature reduction technique in conjunction with a decision tree based classifier to achieve this goal, and the appropriate relevant features were determined. This is what we intend to investigate and implement. Adebowale et al. [9] assessed the performance of well-known attack categorization methods. They concentrated on five of the most common data mining methods used in intrusion detection research: decision trees, Naive Bayes, artificial neural networks, K-nearest neighbour, and support vector machine. They obtained 97 percent accuracy with SVM, 95 percent accuracy with ANN, 99 percent accuracy with KNN, 89 percent accuracy with NB, and 99 percent accuracy with DT. We intend to attain similar accuracy with our new implementations. MeeraGandhi et al. [10] assess the performance of a collection of rule-based and tree-based classifier methods (Decision trees, Random forest, NBTree, REPTree). The best algorithms for each attack type are picked based on the evaluation findings, and two classifier algorithm selection methods are given.

Chapter 4

Dataset

NSL-KDD dataset contains four main files as described in Figure 4.1, and it is a modified version of the KDD-cup99 dataset [12]. It is publicly available for download, and provided by Canadian Institute for Cybersecurity. There are in total 41 features in both training and test sets which are labeled as normal or as an attack of particular type.

File name	Description
KDDTrain+.TXT	Full training set including attack-type labels and difficulty level in CSV format
KDDTest+.TXT	Full test set including attack-type labels and difficulty levels in CSV format
KDDTrain+_20Percent.TXT	A 20% subset of the train set
KDDTest+_20Percent.TXT	A 20% subset of the test set

Figure 4.1: Description of NSL-KDD Dataset

4.1 Features

Among the features present in the data set, the following three types can be cited:

- **Categorical features:** is used to indicate any data represented by strings (see Figure 4.2).

```
In [13]: categorical_cols
Out[13]: ['protocol_type', 'service', 'flag']
```

Figure 4.2: Categorical Features

- **Numerical features:** is used to express any data represented by numbers (as depicted in Figure 4.3), there are a total of 32 features.

```
In [15]: numeric_cols

Out[15]: ['duration',
          'src_bytes',
          'dst_bytes',
          'wrong_fragment',
          'urgent',
          'hot',
          'num_failed_logins',
          'num_compromised',
          'num_root',
          'num_file_creations',
          'num_shells',
          'num_access_files',
          'num_outbound_cmds',
          'count',
          'srv_count',
          'serror_rate',
          'srv_serror_rate',
          'rerror_rate',
          'srv_rerror_rate',
          'same_srv_rate',
          'diff_srv_rate',
          'srv_diff_host_rate',
          'dst_host_count',
          'dst_host_srv_count',
          'dst_host_same_srv_rate',
          'dst_host_diff_srv_rate',
          'dst_host_same_src_port_rate',
          'dst_host_srv_diff_host_rate',
          'dst_host_serror_rate',
          'dst_host_srv_serror_rate',
          'dst_host_rerror_rate',
          'dst_host_srv_rerror_rate']
```

Figure 4.3: Numerical Features

- **Binary features:** is used to designate any data represented by 0 or 1 as seen in Figure 4.4.

```
In [14]: binary_cols

Out[14]: ['land',
          'logged_in',
          'root_shell',
          'su_attempted',
          'is_host_login',
          'is_guest_login']
```

Figure 4.4: Binary Features

4.2 Attack types

The labels of NSL-KDD dataset are mainly categorized into normal and four types of attacks which are DoS, Probe, R2L and U2R.

- **Denial of Service (DoS):** It intends to block or restrict a computer system or network resources or services. There are 6 types of DoS attack in the NSLKDD dataset with total of 45927 instances. The DoS attacks are named as: back (956), land (18), neptune (41,214), pod (201), smurf (2,646), teardrop (892).
- **Probe:** It tries to obtain information from the remote computer by scanning ports. In NSL-KDD dataset, there are 4 types of Probe attacks and total number of samples is 11656. The Probe attacks can be found with the name satan (3,633), ipsweep (3,599), nmap (1,493), portsweep (2,931).
- **Remote to Local (R2L):** The intruder tries to get the unauthorized local access in the victim's computer. There are 9 type R2L attacks are listed in NSL-KDD dataset with total of 1642 instances. The R2L types of attacks are listed as: guess passwd (53), ftp write (8), imap(658), phf (4), multihop (7), warezmaster (20), warezclient (890), spy (2).
- **User to Root (U2R):** In this type of attack the intruder tries to obtain the administrator privileges from the victim computer. There are 4 types of attacks U2R attacks are enlisted in NSL-KDD dataset with total 52 instances. The listed attacks are: buffer overflow (30), loadmodule (9), rootkit (10), perl (3).

Chapter 5

Methodology

Following the same process as any data mining problem, we ran through the following steps.

5.1 Data Preprocessing

The input data is a CSV file with 42 columns. The data was loaded and stored using the Pandas library.

5.1.1 Data Cleaning

First, We began by exploring the binary features. There is one feature that has values of *2.0* which is not expected since binary means *0s* or *1s*. We therefore changed all of the *2.0* values to *0.0* as depicted in Figure 5.1.

```
In [16]: TrainData[binary_cols].describe().transpose()
```

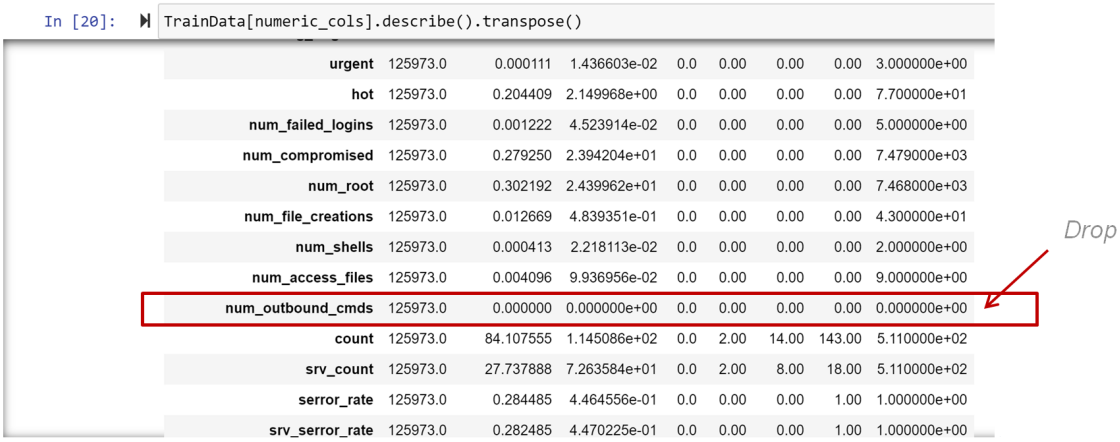
Out[16]:

	count	mean	std	min	25%	50%	75%	max
land	125973.0	0.000198	0.014086	0.0	0.0	0.0	0.0	1.0
logged_in	125973.0	0.395736	0.489010	0.0	0.0	0.0	1.0	1.0
root_shell	125973.0	0.001342	0.036603	0.0	0.0	0.0	0.0	1.0
su_attempted	125973.0	0.001103	0.045154	0.0	0.0	0.0	0.0	2.0
is_host_login	125973.0	0.000008	0.002817	0.0	0.0	0.0	0.0	1.0
is_guest_login	125973.0	0.009423	0.096612	0.0	0.0	0.0	0.0	1.0

Replace by 0.0

Figure 5.1: Binary Features Data Cleaning

Secondly, we investigated the numerical features (Figure 5.2) and discovered that the *num_ outbound _ cmds* have only 0.0 values, so they should be removed.



urgent	125973.0	0.000111	1.436603e-02	0.0	0.00	0.00	0.00	3.000000e+00
hot	125973.0	0.204409	2.149968e+00	0.0	0.00	0.00	0.00	7.700000e+01
num_failed_logins	125973.0	0.001222	4.523914e-02	0.0	0.00	0.00	0.00	5.000000e+00
num_compromised	125973.0	0.279250	2.394204e+01	0.0	0.00	0.00	0.00	7.479000e+03
num_root	125973.0	0.302192	2.439962e+01	0.0	0.00	0.00	0.00	7.468000e+03
num_file_creations	125973.0	0.012669	4.839351e-01	0.0	0.00	0.00	0.00	4.300000e+01
num_shells	125973.0	0.000413	2.218113e-02	0.0	0.00	0.00	0.00	2.000000e+00
num_access_files	125973.0	0.004096	9.936956e-02	0.0	0.00	0.00	0.00	9.000000e+00
num_outbound_cmds	125973.0	0.000000	0.000000e+00	0.0	0.00	0.00	0.00	0.000000e+00
count	125973.0	84.107555	1.145086e+02	0.0	2.00	14.00	143.00	5.110000e+02
srv_count	125973.0	27.737888	7.263584e+01	0.0	2.00	8.00	18.00	5.110000e+02
serror_rate	125973.0	0.284485	4.464556e-01	0.0	0.00	0.00	1.00	1.000000e+00
srv_serror_rate	125973.0	0.282485	4.470225e-01	0.0	0.00	0.00	1.00	1.000000e+00

Figure 5.2: Numerical Features Data Cleaning

In the last step, we examined the categorical features shown in Figure 5.3, and we encountered two problems. Since data should be compatible, We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information. Encoding is a required pre-processing step when working with categorical data for machine learning algorithms. In our case, we used ***OneHotEncoding***, which is an encoding technique that has the capability to transform the categorical features into binary features. It creates new binary columns, indicating the presence of each possible value from the original data. Another challenge is to balance the number of features in the train and test data. If we look at the number of distinct values for the *service* feature, we can see that there are six additional values in the train data (see Figure 5.3).

Figure 5.3: Features Balancing

Due to the use of ***OneHotEncoding***, the number of columns in the train will be greater than the number of columns in the test (see Figure 5.4 and Figure 5.5). Consequently, we added

the six missing columns in the test data so that we could fit and predict when we created the model (see Figure 5.6).

```
In [29]: TrainData.head(5)
```

Out[29]:

	duration	src_bytes	dst_bytes	land	wrong_fragment
0	0	491	0	0	0
1	0	146	0	0	0
2	0	0	0	0	0
3	0	232	8153	0	0
4	0	199	420	0	0

5 rows × 122 columns

Figure 5.4: Train Data Features

```
In [30]: TestData.head(5)
```

Out[30]:

	duration	src_bytes	dst_bytes	land	wrong_fragment
0	0	0	0	0	0
1	0	0	0	0	0
2	2	12983	0	0	0
3	0	20	0	0	0
4	1	0	15	0	0

5 rows × 116 columns

Figure 5.5: Test Data Features Before

```
In [35]: TestData.head(5)
```

Out[35]:

	duration	src_bytes	dst_bytes	land	wrong_fragment
0	0	0	0	0	0
1	0	0	0	0	0
2	2	12983	0	0	0
3	0	20	0	0	0
4	1	0	15	0	0

Adding the missing columns

5 rows × 122 columns

Figure 5.6: Test Data Features After

5.1.2 Feature Scaling

Features Scaling is a common requirement of machine learning methods, to avoid that features with large values may weight too much on the final results. For each feature, calculate the average, subtract the mean value from the feature value, and divide the result by their standard deviation. After scaling, each feature will have a zero average, with a standard deviation of one (as seen in Figure 5.7).

```
In [43]:  print(sc_data.std(axis=0))

[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1.]
```

Figure 5.7: Features Scaling

5.2 Binary and Multi-class Problems

One of our contributions is to build two models, one solving a binary problem and the other solving a multiclass problem. We used the *Label Encoder* to convert the labels into numerical values. For the binary problem, we only have two classes *normal*=0 & *attack*=1. The attack class can be of one type, so we divided our dataset into 4 sub datasets. For the multiclass problem, we have five classes distributed as follows *normal*=0, *DoS*=1, *Probe*=2, *R2L*=3, *U2R*=4

5.3 Data Augmentation

Data Augmentation utilizes existing data to create more training sets by generating changed data from the original set. When learning a model, it's a wise idea to apply this technique to avoid over-fitting when we have small dataset, so we have to ensure that the dataset is large enough we want the model to perform well. We explored three different DA techniques:

- **UnderSampling:** We will not benefit from this approach because it will reduce the number of instances of the majority classes until we reach the minority classes, and the minority classes U2R are too small, which will lead to overfitting.
- **OverSampling:** It is also beneficial because in a way, it duplicates the instances of the minority classes to reach the majority class.
- **SMOTE** It is better to use this method because it increases the number of instances without duplication of the instances of the minority class. (see Figure 5.8 and Figure 5.9)

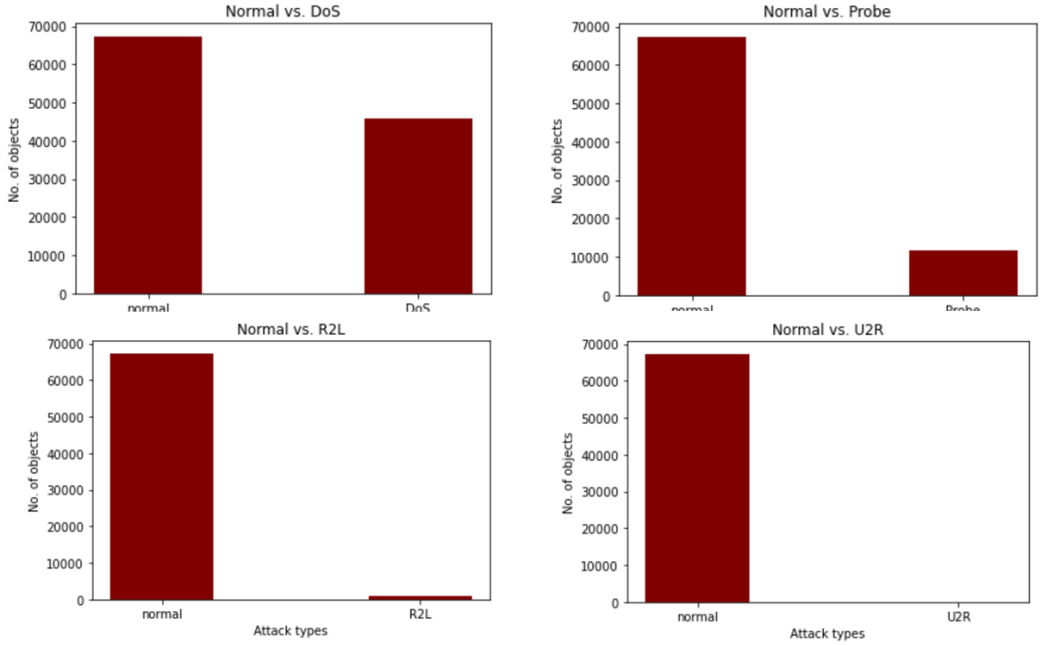


Figure 5.8: Unbalanced Data

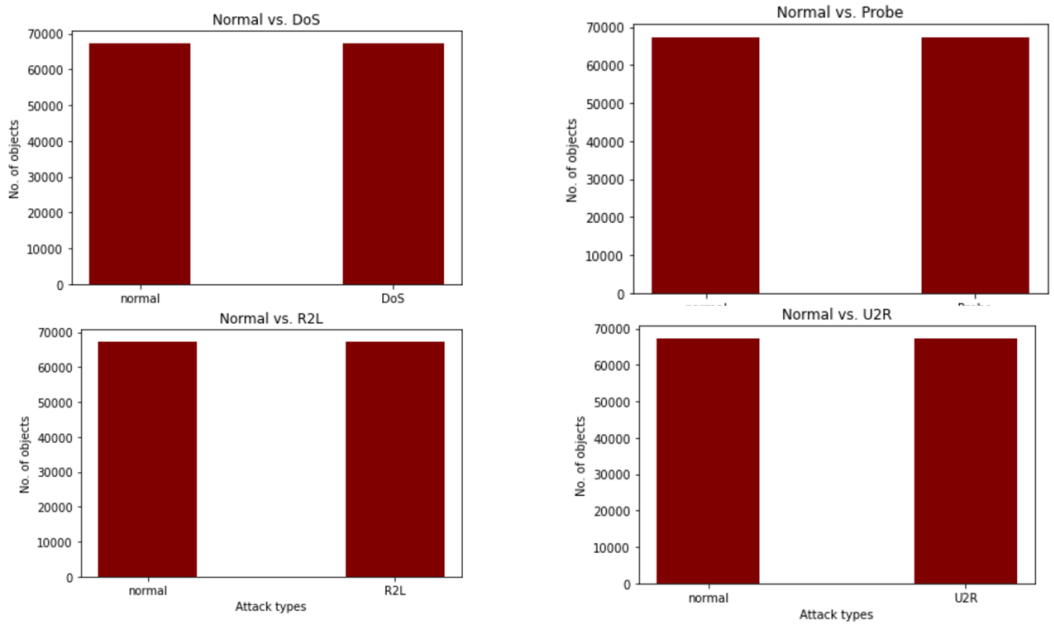


Figure 5.9: SMOTE Technique

5.4 Feature Selection

When creating a predictive model, feature selection is the process of minimizing the number of input variables. It is preferable to limit the number of input variables in order to reduce modelling computational costs and, in certain situations, increase model performance. In our project, we used the SelectFromModel technique, however, what we did as a proposed process is to run 4 different algorithms (Fig 5.10) to select the effective features in our dataset, then we choose the features that are present in at least on of our used algorithms as our final selected features as depicted in Figure 5.11.

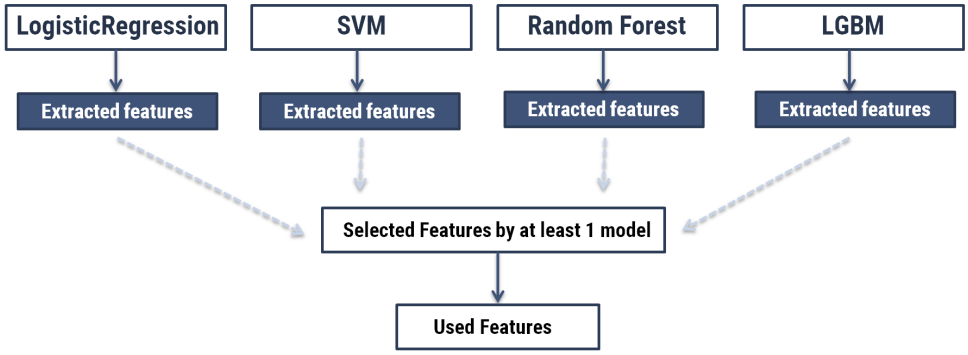


Figure 5.10: Features Extraction

Out[107]:

	Feature	Logistics	Random Forest	svm	LightGBM	Total
1	srv_count	True	True	True	True	4
2	service_19	True	True	True	True	4
3	same_srv_rate	True	True	True	True	4
4	protocol_type_1	True	True	True	True	4
5	logged_in	True	True	True	True	4
6	hot	True	True	True	True	4
7	duration	True	True	True	True	4
8	dst_host_srv_serror_rate	True	True	True	True	4
9	dst_host_srv_diff_host_rate	True	True	True	True	4
10	dst_host_srv_count	True	True	True	True	4
11	dst_host_serror_rate	True	True	True	True	4
12	dst_host_same_srv_rate	True	True	True	True	4

Figure 5.11: Feature Extraction Result

5.5 Models

In data mining (or machine learning), an algorithm is a combination of heuristics and computations that generates a model from data. The algorithm initially examines the data you submit to develop a model, looking for particular sorts of patterns or trends. The algorithm employs the findings of this study over a number of iterations to determine the best parameters for developing the mining model. These parameters are then applied to the full data collection in order to discover actionable patterns and statistics. In our analysis, we implemented the below algorithms in python from scratch.

5.5.1 Naive Bayes

It is a classification strategy based on Bayes' Theorem and the assumption of predictor independence. A Naive Bayes classifier, in basic words, posits that the existence of one feature in a class is independent to the presence of any other feature. The Naive Bayes model is simple to construct and is especially good for very big data sets. In addition to its simplicity, Naive Bayes has been shown to beat even the most advanced classification systems. [2]. A Bayesian classifier is based on the premise that if an agent knows the class, it can anticipate the values of the other attributes. If it does not know the class, it can apply Bayes' rule to guess the class based on (part of) the feature values. The learning agent in a Bayesian classifier constructs a probabilistic model of the characteristics and utilises that model to predict the classification of a new example. A latent variable is a probabilistic variable that goes unnoticed. A Bayesian classifier is a probabilistic model in which the classification is a latent variable that is connected to the observable variables in a probabilistic way. In the probabilistic model, classification becomes inference.[3] Since all characteristics are assumed to be independent, the naive bayes algorithm is particularly quick when compared to sophisticated algorithms. In certain circumstances, speed above precision is desirable. It is effective with high-dimensional data, such as text categorization and email spam detection. Because the assumption that all characteristics are independent is seldom true in practise, the naive bayes algorithm is less accurate than more elaborate algorithms. Speed comes at a price! [4]

5.5.2 k-Nearest Neighbors

The K-nearest neighbours (KNN) algorithm belongs to the category of supervised machine learning methods. In its most basic form, KNN is relatively simple to create while performing quite difficult classification jobs. It is a lazy learning algorithm since it lacks a dedicated training step. Rather, while categorising a new data point or instance, it uses all of the data for training. KNN is a non-parametric learning algorithm, which means it makes no assumptions about the data. This is a very helpful characteristic because most real-world data does not adhere to any theoretical assumptions, such as linear separability, uniform distribution, and so on. It is really simple to implement. As previously said, it is a lazy learning system that requires no training before producing real-time predictions. As a result, the KNN method is significantly faster than other algorithms that need training, such as SVM, linear regression, and so on. Since the system does not require any training before producing predictions, fresh data may be simply integrated. To implement KNN, just two parameters are required: the value of K and the distance function (e.g. Euclidean or Manhattan etc.). The KNN technique does not perform well with high-dimensional data because it becomes harder for the algorithm to compute distance in each dimension as the number of dimensions increases. For big datasets, the KNN method has a significant prediction cost. This is due to the increased expense of computing distance between each new point and

each old point in huge datasets. Finally, the KNN technique does not function well with categorical data since finding the distance between dimensions using categorical characteristics is challenging. [7]

Chapter 6

Results

6.1 Multiclass

Analysis of the multiclass problem (Fig 6.1) shows that KNN has the highest accuracy, so it performs better than NB. However, accuracy can be misleading from time to time. Thus, we must check other metrics like precision, recall, and F1-measure.
Precision: measures how many of all elements that the model classified as positive were in fact correct.
Recall: is a metric used to explain the balance between the number of instances classified as positive by the model compared to the number of instances classified as negative but they are in reality positive.
F1-meaure: which is the harmonic mean or ratio between precision and recall.
NB performs the worst because it has the lowest F-1 measure, so KNN performs the best since it has the highest F1-measure.

Model	Accuracy	Precision	Recall	F1-measure
KNN	0.97	0.962	0.89	0.915
NB	0.53	0.133	0.25	0.173

Figure 6.1: Multi-class Problem Results

6.2 Binary

As can be seen from Fig. In 6.2, we show the classifiers that perform the most effectively on each attack type. KNN outperforms NB in all type of attacks, We can also see how much gain we get for the R2L and U2R attacks, which were the minority attacks, after applying the SMOTE DA algorithm.

KNN - DoS

	precision	recall	f1-score
0.0	0.87	0.99	0.93
1.0	0.99	0.80	0.88
accuracy			0.91
macro avg	0.93	0.90	0.90
weighted avg	0.92	0.91	0.91

KNN - Probe

	precision	recall	f1-score
0.0	0.80	0.87	0.84
1.0	0.89	0.84	0.87
accuracy			0.85
macro avg	0.85	0.85	0.85
weighted avg	0.86	0.85	0.85

KNN – R2L

	precision	recall	f1-score
0.0	0.74	0.99	0.85
1.0	0.99	0.60	0.74
accuracy			0.81
macro avg	0.86	0.80	0.79
weighted avg	0.85	0.81	0.80

KNN – U2R

	precision	recall	f1-score
0.0	0.61	0.97	0.75
1.0	0.96	0.54	0.69
accuracy			0.72
macro avg	0.79	0.75	0.72
weighted avg	0.81	0.72	0.72

Figure 6.2: Binary-class Problem Results

When comparing the Binary and Multiclass, we recommend to use the Multiclass model as it has a better performance and better knowledge of explaining and detecting the attacks.

Chapter 7

Conclusion

To sum up, in our project we implemented a novel ML-based IDS that can be used for binary and multi-class problems. The main distinction of our system is its ability to significantly decrease the false negative alarms which is one of the problems that most conventional intrusion detection systems suffer from. Moreover, our ML-based IDS implements a new feature extraction technique by taking the features that are extracted from each SelectFromModel method. Additionaly, we balanced our data using SMOTE data augmentation technique which helped us to avoid overfitting and getting better model's performance. Our experiments validate the efficiency of our system. Our results justify that KNN outperforms other algorithms in best classifying the attacks, and on the other side Naive Bayes algorithm seems to be the less accurate algorithm which is not surprising as it assumes that all the characteristics are independent which is not true in real life.

Chapter 8

REFERENCES

[1] https://towardsai.net/p/machine-learning/why-choose-random-forest-and-not-decision-trees
[2] https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
[3] https://artint.info/html/ArtInt_181.html
[4] https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed
[5] https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
[6] https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications/
[7] https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/ [8] Nkiama, Herve & Zainudeen, Syed & Saidu, Muhammad. (2016). A Subset Feature, Elimination Mechanism for Intrusion Detection System. International Journal of Advanced Computer Science and Applications. 7. 10.14569/IJACSA.2016.070419
[9] Ajayi, Adebowale & S.A, Idowu. (2013). Comparative study of selected data mining algorithms used for intrusion detection, IJCSE, 2013.
[10] Dhanabal, L., Dr. S.P. Shantharajah, "A Study on NSL_KDD Dataset for Intrusion Detection System Based on Classification Algorithms," International Journal of Advanced Research in Computer and Communication Engineering, vol. 4, issue 6, pp. 446-452, June 2015
[11] https://dhirajkumarblog.medium.com/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a
[12] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In 2009 IEEE symposium on computational intelligence for security and defense applications, pages 1–6. IEEE, 2009.