HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

APPLIED STATISTICS AND EXPERIMENTAL DESIGN

PROJECT: NETWORK ATTACKS DETECTION

Instructor: Assoc Prof. Linh Giang Nguyen

Students: Nguyen Quang Duc - 20204876
La Dai Lam - 20204918
Le Hong Duc - 20204874
Luu Trong Nghia - 20204888
Tran Hoang Quoc Anh - 20200044

Ha Noi, July - 2022

Contents

1 Introduction

Internet is not a new concept to everyone, it is a global system of interconnected computer networks, and everyone participates in data traffic between each other. However, for any web server, there is always a chance of getting attacked, whether by DDOS, Website Defacement, Directory Traversal, etc. To ensure secure, reliable, and qualitative network communication, we need a good way to analyze and supervise our traffic. Several models and methods have been proposed and implemented, and some attain desired efficiency and results to determine whether or not a connection is an attack or a normal one.

In this project, our group will try to build software to detect network intrusions protect a computer network from unauthorized users, including perhaps insiders. The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections.[1]

Our report first will introduce the datasets we have chosen which is KDD Cup 1999 Dataset.[1] The following sections are exploratory data analysis (EDA) which we explored the data features, and patterns and then we tried different models for the attacks detection problem with suitable methods and evaluations. Finally, we will summarize what we have found out and some of the important points for this problem.

2 Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is one of the most important steps before doing anything in every tabular dataset problem. As we used KDD Cup 1999 Dataset, EDA helps us gain insight into the dataset by visualizing charts and detecting any errors, and outliers as well as understanding different data patterns of many kinds of network attacks. In this section, we tried to depict what we have done and how we preprocessed the dataset.

2.1 Data Understanding & Visualization

2.1.1 Datasets

Firstly, we will give a brief description of our dataset - KDD Cup 99.[1] Since 1999, it has been the most wildly used data set for the evaluation of anomaly detection (network attack detection) methods.[2] KDD training set contains nearly 4,900,000 instances (single connection between two computers) but we just selected 10 percent of the training set for training and evaluating our machine learning system. Every single connection is represented by 41 features and is labeled with exactly one specific type of attacks. The simulated attacks are categorized into one of four categories:

- Denial of Service Attack (DoS)
- User to Root Attack (U2R)
- Remote to Local Attack (R2L)
- Probing Attack

Besides the target variable, KDD Cup 99 features can be classified into three groups (two derived feature categories):

- **Basic features**: This category contains all features which can be extracted directly from the TCP/IP connections which are depicted in Figure 1.
- **Content features**: This category includes all features which can be inferred by the domain knowledge of experts which helps us to classify better (Figure 2).
- **Traffic features**: This category contains all features which could be calculated by using a two-second time window (before the current connection) (Figure 3)

feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Figure 1: Basic features of individual TCP connections

feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

Figure 2: Content features within a connection suggested by domain knowledge

2.1.2 Data Exploration

2.1.2.a Univariate Analysis

Firstly, we will plot all the histograms of all numeric attributes in order to give us a brief information about the distribution of some numeric features as depicted in Figure 4.

Overall, numeric features in KDD Cup 99 are severely suffered from skewness as you can see in Figure 4, almost features have one dominant value such as (duration, src_bytes,dst_bytes). Therefore, we thought that maybe some outliers of those attributes would indicate which connections are bad.

After that, we will also plot all histograms of all categorical attributes including target, flag, service, and protocol_type as depicted in Figure 5.

In general, all connections fall into three dominant types (smurf, neptune, and normal) as well as ecr_i, private, and http are the most popular service features in the dataset.

Finally, we will also take account into the distribution of our target label which is 'Attack Type' in Figure 6. As we can see, the almost connection is labeled as 'dos' which accounts for nearly 80%. Further more, 19.6% belongs to normal connection and another small percentage falls into other attack types.

2.1.2.b Multivariate Analysis

In this part, we will analyze the relationship between some independent features with dependent

feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Figure 3: Traffic features computed using a two-second time window

features as well as the correlation between each independent features which would help us tremendously at data preparation stage.

Our first step is finding the relationship between some independent features (protocol_type, is_guest_login) towards the target attribute ('Attack Type') as depicted in Figure 7. For the influence of protocol type, udp protocol_type will tend to have normal connections whereas the connection with tcmp will tend to be dos. So we decided to keep this feature rather than drop it.

Next, we plot the correlation between independent variables as in Figure 8. In general, there are many high correlation between some pair of variables such as (num_root, num_compromised), (srv_serror_rate, serror_rate), (dst_host_serror_rate, serror_rate), etc. which will help us to remove some redundant variables in the dataset.

2.2 Data Preparation

Data preparation is a pre-processing step that involves cleansing, transforming, and consolidating data.[3] In this part, we will illustrate how we preprocessed our dataset based on the previous section (EDA).

2.2.1 Data Cleaning

In the first step, we would like to check whether there is a missing value in each feature of in the dataset. Therefore, we depicted the heatmap of null values for each attribute in Figure 9. As we can see, there is no null values for all attributes so we decided to not drop a feature or delete any instances yet.

2.2.2 Redundant Variables

In this step, we would like to remove redundant variables where they have only one unique values such as (num_outbound_cmds, is_host_login) and some variables which have a high correlation value with other variables such as (num_root, srv_serror_rate, etc.) which based on Figure 8.

After doing two steps, our training set has 33 features (less than 8 features than the raw dataset).

2.2.3 Variable Transformations

Before training the model, we need to transform text and categorical to numeric values. In order to that, we chose label encoder and standard scalar (standardize) for categorical and numeric values, respectively.

So now our dataset is ready for training and predicting.

Figure 4: Distribution of numeric features

3 Modelling

3.1 Probabilistic models

In this section, we want to mention about probabilistic models first. This is because the majority of this project was about statistical analysis therefore we were more likely to apply models that involve uncertainty before the others.

3.1.1 Gaussian Naive Bayes

Before diving into Gaussian Naive Bayes, we will mention about the overview of Naive Bayes methods. These are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n, :

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

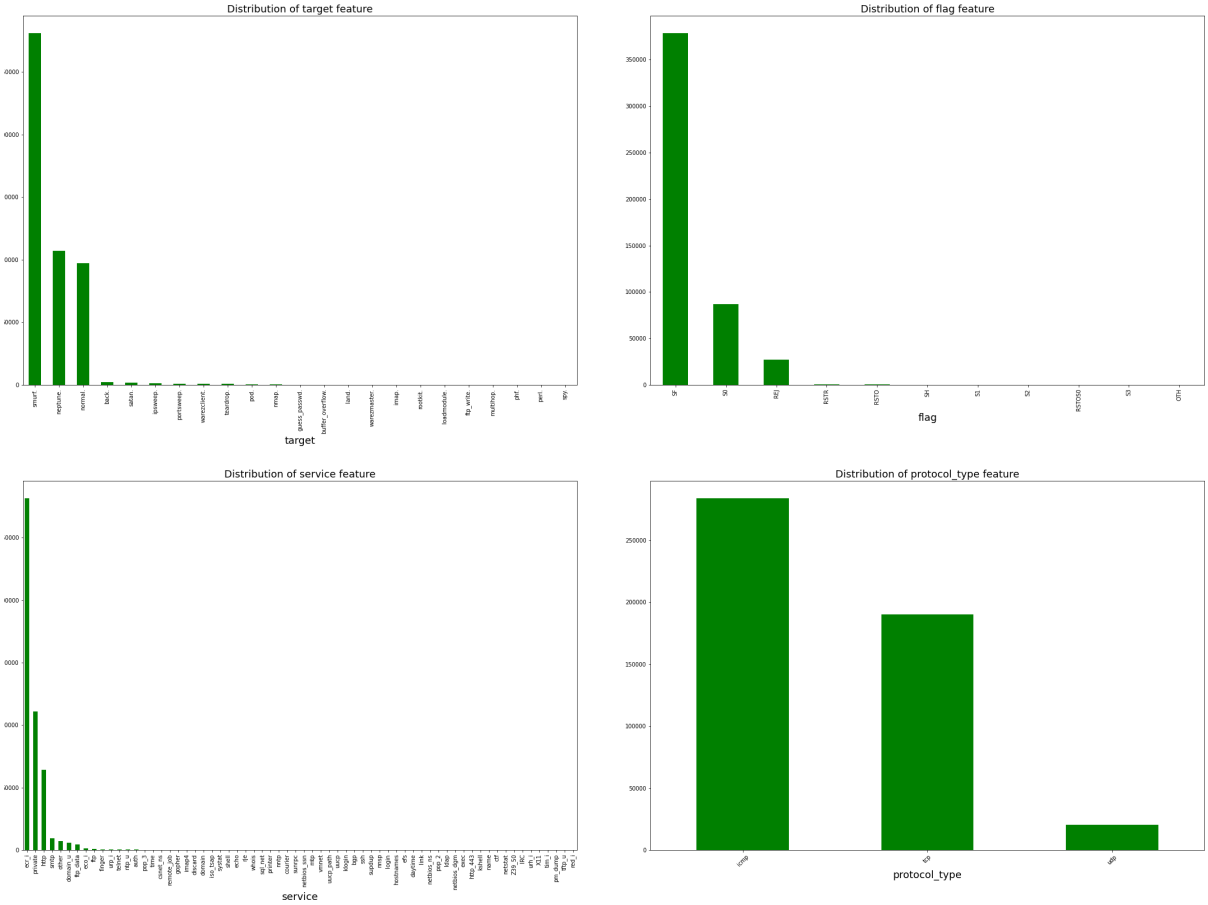

Figure 5: Distribution of categorical features

Using the naive conditional independence assumption that:

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$$

for all i, this relationship is simplified to:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a Normal Distribution (or Gaussian Distribution). The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

An approach to creating a simple model is to assume that the data is described by a Gaussian distribution with no co-variance (independent dimensions) between dimensions. This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all that is needed to define such a distribution.

3.1.2 Multinomial Naive Bayes

While Gaussian Naive Bayes supports continuous-valued features, the Multinomial Naive Bayes classifier is suitable for classification with discrete features. This algorithm is one of the two classic naive Bayes variants used in text classification. The multinomial distribution normally requires integer feature counts. The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the

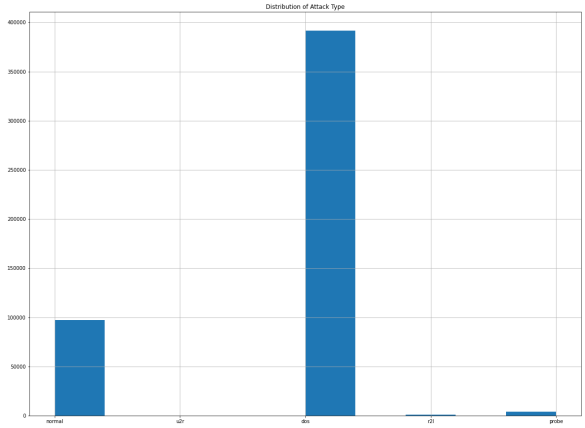

Figure 6: Distribution of target feature - 'Attack Type'

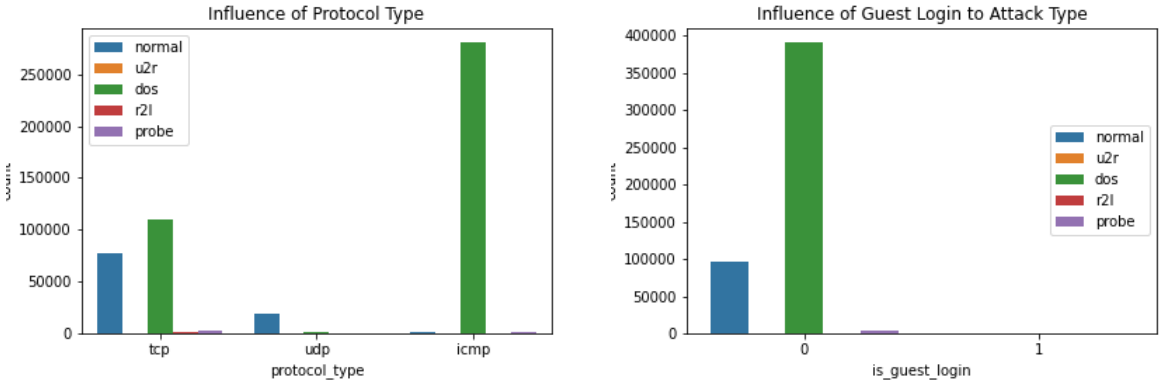

Figure 7: Influence of protocol type and attack type towards dependent variable

number of features and θ_{yi} is the probability $P(x_i|y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ account for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone Smoothing.

3.1.3 Gaussian Mixture Model

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussian Models.

A Gaussian mixture model is parameterized by two types of values, the mixture **component weights** and the component **means** and **variances/covariances**. For a Gaussian mixture model with K component, the k^{th} component has a mean of μ_k and variance of σ_k.

If the number of components K is known, **expectation maximization** is the technique most commonly used to estimate the mixture model's parameters. In Frequentist Probability theory, models are typically learned by using maximum likelihood estimation techniques, which seek to maximize the probability, or likelihood, of the observed data given the model parameters. Unfortunately, finding the maximum likelihood solution for mixture models by differentiating the log likelihood and solving for 0 is usually analytically impossible.

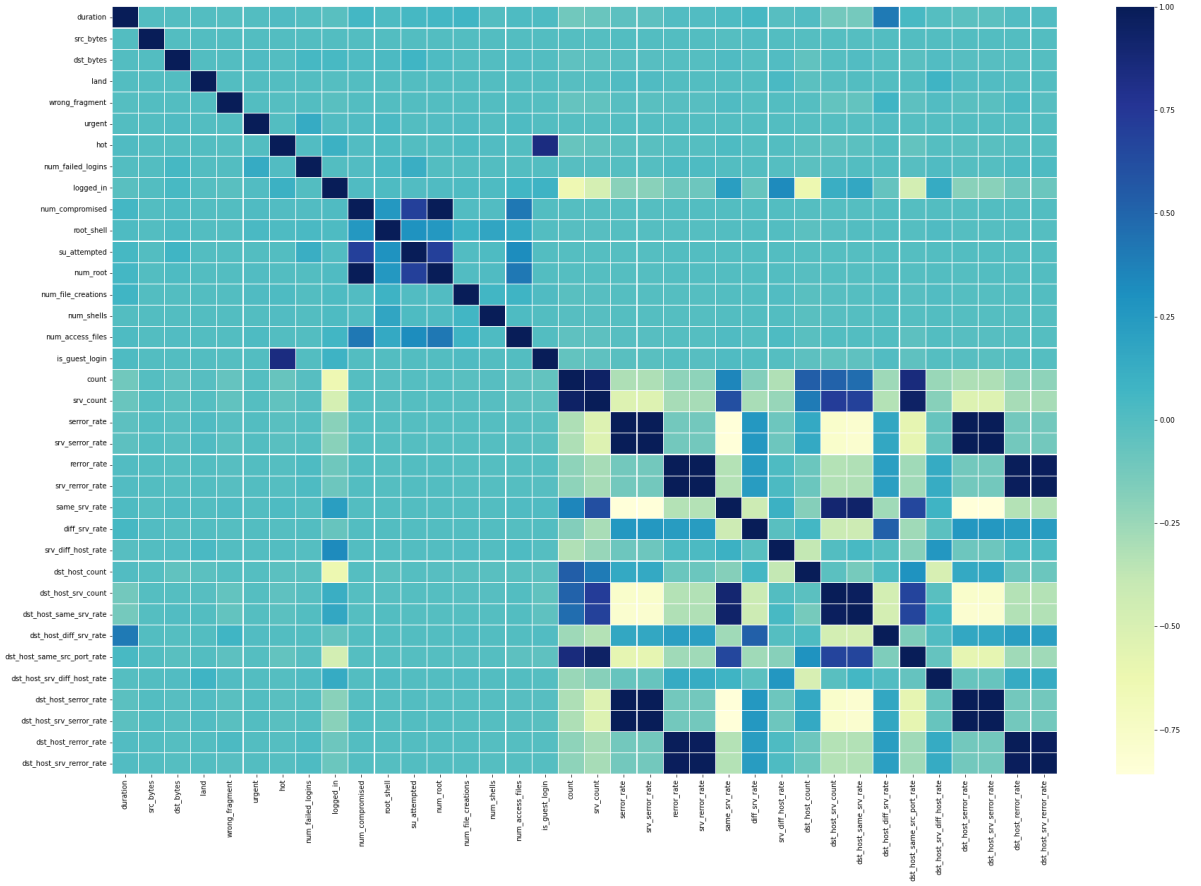

Figure 8: The heatmap representing correlation of independent variables

Expectation maximization for mixture models consists of two steps.

1. The first step, known as the expectation step or E step, consists of calculating the expectation of the component assignments C_k for each data point $x_i \in X$ given the model parameters ϕ_k, μ_k , and σ_k.

2. The second step is known as the maximization step or M step, which consists of maximizing the expectations calculated in the E step with respect to the model parameters. This step consists of updating the values ϕ_k, μ_k , and σ_k.

When the number of components K is not known a priori, it is typical to guess the number of components and fit that model to the data using the EM algorithm. This is done for many different values of K. Usually, the model with the best trade-off between fit and number of components is kept.

3.2 Other Machine Learning Models

We also want to apply some other Machine Learning Models for comparing and improving results. In the next section (3.2.1), we will remark on Logistic Regression theory; in section 3.2.2, we will demonstrate how Support Vector Machine works. Section 3.2.3 and 3.2.4 give information of Decision Tree and Random Forest Models, respectively. After that, in section 3.2.5, we will mention about AdaBoost Algorithm.

3.2.1 Logistic Regression

Logistic Regression is an example of a classification algorithm which is used to find a relationship between features and the probability of a particular outcome. "Logistic" is taken from the Logit Function that is used in this method of classification. The term "Regression" comes from its underlying technique which is quite the same as Linear regression. However, unlike regular regression, the outcome calculates the predicted probability of mutually exclusive events occurring based on multiple external factors. Now, we will focus more on the Logit Function.

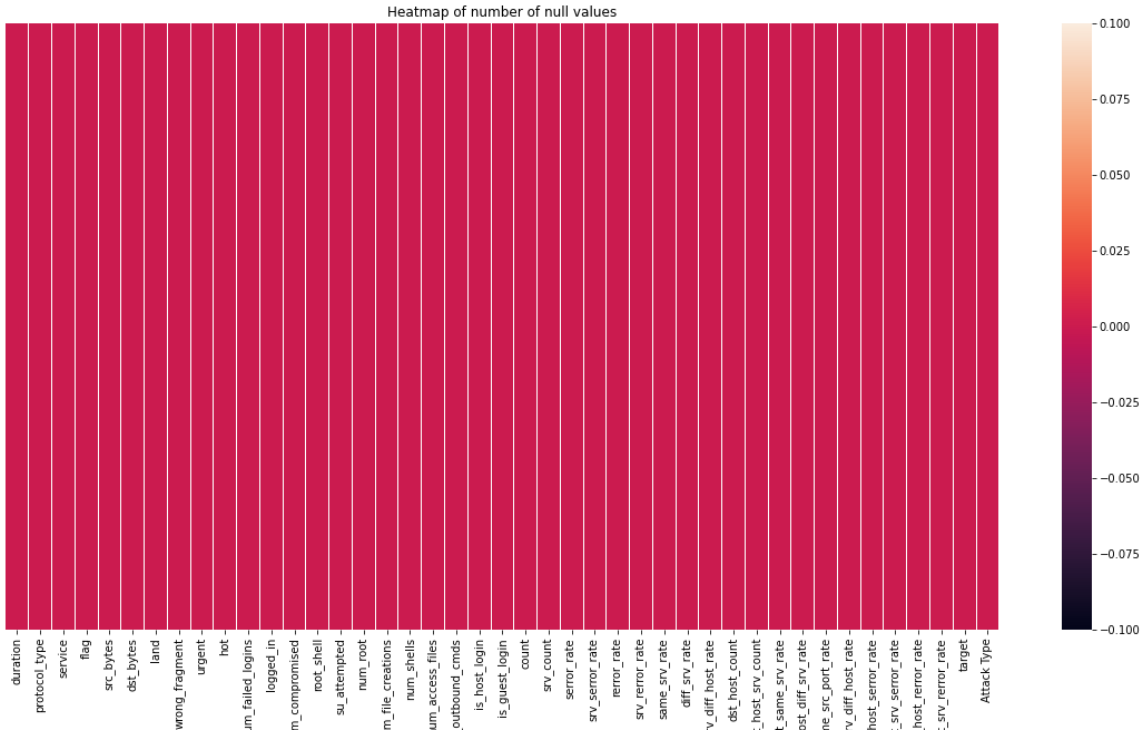

Figure 9: The heatmap representing the number of null values

We have the linear regression equation:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

Here, y is dependent variable and $X_1, X_2, \ldots X_n$ are explanatory variables.

The sigmoid function can be written as:

$$P = \frac{1}{1 + e^{-y}}$$

The sigmoid function gives an S-shaped curve. It always gives a value of probability ranging from $0 < P < 1$. The function can take any real-valued number and map it into a value between 0 and 1. If we apply the sigmoid equation to the linear regression equation, we have the equation for Logistic Regression:

$$P = \frac{e^{(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}{1 + e^{(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}$$

Logistic regression can be expressed as:

$$\log(\frac{p(X)}{1 - p(X)}) = \beta_0 + \beta_1 X$$

where the left-hand side is called the logit or log-odds function, and p(x)/(1-p(x)) is called odds.

The odds signify the ratio of the probability of success to the probability of failure. Therefore, in Logistic Regression, the linear combination of inputs is mapped to the log (odds) which is the output being equal to 1.

3.2.2 Support Vector Machine

Support Vector Machine or SVM is a linear model for classification and regression problem. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes. From now, we present mainly linear classification.

Linear separability assumption: there exists a hyperplane (of linear form) that well separates the two classes.

SVM finds a hyperplane of the form:

$$f(x) =< w \cdot x > +b$$

where w is the weight vector, b is a real number (bias), $< w \cdot x >$ or $< w, x >$ denote the inner product of two vectors.

For each x_i, we have:

$$y_i = \begin{cases} 1 \text{ if } < w, x_i > +b \geq 0 \\ -1 \text{ if } < w, x_i > +b < 0 \end{cases}$$

The hyperplane (H_0) which separates the positive from negative class is of the form $< w, x > +b = 0$. This equation is also known as the *decision boundary/surface*. But in practice, there will be infinitely many separating hyperplanes satisfy the decision boundary equation. Then, SVM will selects the hyperplane with max margin.

We define two parallel marginal hyperplanes as follows:

- H_+ crosses x_+ and is parallel with H_0: $< w, x^* > +b = 1$
- H_- crosses x_- and is parallel with H_0: $< w, x^* > +b = -1$
- $(x_+, 1)$ in positive class and $(x_-, 1)$ in negative class which are closest to the separating hyperplane H_0.

Margin is defined as the distance between the two marginal hyperplanes. $(d_+ + d_-)$ is the margin where d_+ be the distance from H_0 to H_+, d_- be the distance from H_0 to H_-.

For many problems, our linear separability assumption does not hold, we cannot apply Linear SVM as usual, but the idea of Non-linear SVM solve this:

- Step 1: Transform the input into another space, which often has higher dimensions, so that the projection of data is linearly separable.
- Step 2: Use Linear SVM in the new space.

3.2.3 Decision Tree

A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered. The model is a form of supervised learning, meaning that the model is trained and tested on a data set containing the desired categorization.

There are some key terms of a decision tree:

- Root node: The base of the decision tree.
- Splitting: The process of dividing a node into multiple sub-nodes.
- Decision node: When a sub-node is further split into additional sub-nodes.
- Leaf node: When a sub-node does not further split into additional sub-nodes; it represents possible outcomes.
- Pruning: The process of removing sub-nodes of a decision tree.
- Branch: A subsection of the decision tree consisting of multiple nodes.

Before entering the Decision Tree algorithm, some new concepts need to be considered: Entropy and Information Gain.

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Mathematically Entropy for one attribute is represented as:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Mathematically Entropy for multiple attributes is represented as:

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification.

$$\text{Information Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

Now, Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The algorithm selection is also based on the type of target variables. In this project, we only mention about ID3 algorithm (extension of D3). Steps in ID3 algorithm:

1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Entropy and Information Gain of this attribute.
3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

The common problem with Decision trees, especially having a table full of columns, they fit a lot. f there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Here are two ways to remove overfitting:

1. Pruning Decision Trees
2. Stop learning early

3.2.4 Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees. The logic behind the Random Forest model is these multiple uncorrelated models (the individual decision trees) perform much better as a group than they do alone. When using Random Forest for classification, each tree gives a classification or a vote. The forest will choose the classification with the majority of the votes.

Any Random Forest Model has these three basic ingredients. The first ingredient is **randomization and no pruning**. For each tree and at each node, we select randomly a subset of attributes, find the best split, and then grow appropriate its subtrees. Pruning is a technique to avoid overfitting of Decision Tree models, but this technique will not be used in Random Forest models, every tree will be grown to its largest size. The next is **combination**. Each prediction later is made by taking the average of all predictions of individual trees. In more detail, when using Random Forest for regression, the forest picks the average of the outputs of all trees. In classification, each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. Finally, Random Forest is usually trained using the **bagging** method. The bagging method is a type of ensemble machine learning algorithm called Bootstrap Aggregation. Using bagging method, the training set for each tree is generated by sampling (with replacement) from the original data.

Random Forest can be implemented easily and efficiently. Figure Figure **??** illustrates one of many ways to implement Random Forest. One more thing is that Random Forest also can work with problems of very high dimensions, without overfitting.

3.2.5 AdaBoost

Similar to Random Forest Classifier, Adaboost is also an ensemble classifier. Adaboost classifier combines weak classifier algorithm to form strong classifier. A single algorithm may classify the objects poorly. But if we combine multiple classifiers with selection of training set at every iteration and assigning right amount of weight in final voting, we can have good accuracy score for overall classifier.

Basically, Adaboost,

1. Retrains the algorithm iteratively by choosing the training set based on accuracy of previous training.
2. The weight-age of each trained classifier at any iteration depends on the accuracy achieved.

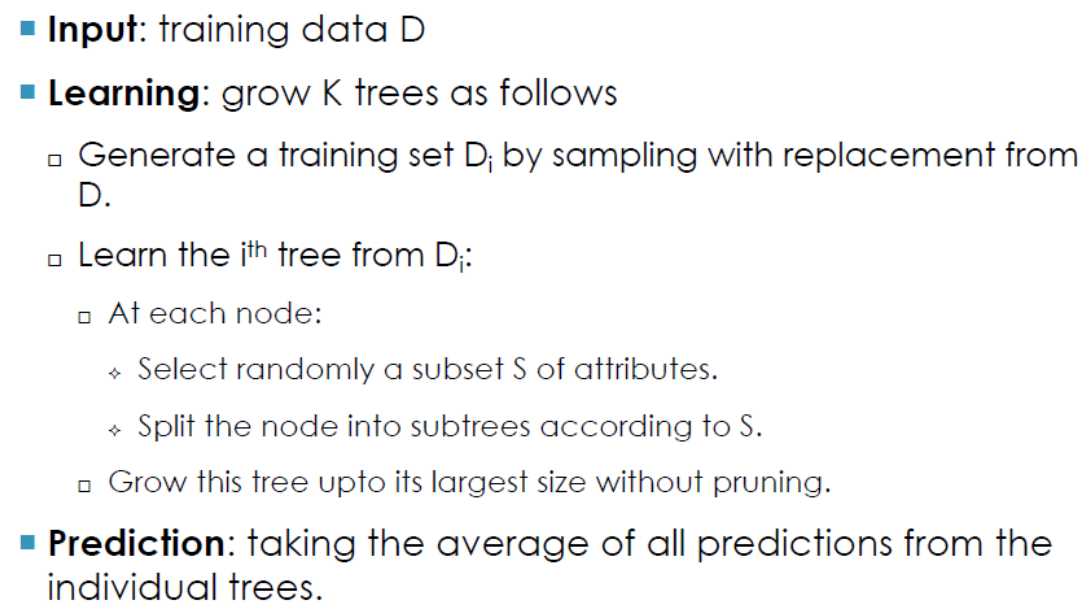

Figure 10: Algorithm of Random Forest

Each weak classifier is trained using a **random subset** of overall training set, but random subset is not 100% random. After training a classifier at any level, Adaboost assigns weight to each training item. Misclassified item is assigned higher weight so that it appears in the training subset of next classifier with higher probability. After each classifier is trained, the weight is assigned to the classifier as well based on accuracy. More accurate classifier is assigned higher weight so that it will have more impact in final outcome.

A classifier with 50% accuracy is given a weight of zero, and a classifier with less than 50% accuracy is given negative weight.

Let's look at the mathematical formula and parameters

$$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$$

where $h_t(x)$ is the output of weak classifier t for input x, α_t is calculates by:

$$\alpha_t = \frac{1}{2} \ln(\frac{1-E}{E}), \text{ E is the error rate}$$

Initially, all the input training example has equal weightage. After weak classifier is trained, we update the weight of each training example with following formula:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

D_t is weight at previous level. We normalize the weights by dividing each of them by the sum of all the weights, Z_t.

In the case of incorrect classification, the exponential term became larger than 1, and in the case of correct classification, the exponential term became below 1. Therefore, incorrect classification would receive higher weights, prompting our classifiers to pay more attention to them in the next iteration, while the opposite case of correct classification would result in the converse.

We continue the iteration until training error achieved is low enough. We then take the final prediction by adding up the weighted prediction of every classifier.

4 Practical Result

4.1 Evaluation Metrics

In our project,in order to evaluate the effectiveness of the models we tried to use different metrics[4] for evaluating each model rigorously such as Accuracy score, Precision score, Recall score, F1-score because of their distinctive advantages:

- Accuracy score: The metric allows us to have a big picture of the correct percentage in our prediction. However, it would be inappropriate for imbalanced-class distributions[5] like this problem.
- Precision score: The metric summarizes the fraction of examples assigned to the positive class that belongs to the positive class.
- Recall score: The metric summarizes how well the positive class was predicted over the total number of actual instances of this class.
- F1-score: The metric is a combination of Precision and Recall metrics, which seeks the balance of both concerns.
- Macro average accuracy/precision/recall score: The metric is measured by taking an average of all same metrics for each label class which would help us to identify whether class labels are biased or not.

As our project is about network attacks detection so we need to have a specific concern about recall score for some "bad" labels as it is much more important to prevent defective connections from computers to our server.

4.2 Result Table

By using scikit-learn[6] library, we have implemented all models and above methods and got the follow results

Model	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-score
Gaussian Naive Bayes	0.89	0.50	0.78	0.48
Multinomial Naive Bayes	0.98	0.75	0.68	0.70
Gaussian Mixture Model	0.57	0.20	0.16	0.17
Logistic Regression	0.96	0.53	0.67	0.53
Support Vector Machine	1.00	0.94	0.89	0.92
Decision Tree	0.99	0.51	0.58	0.54
Random Forest	1.00	0.98	0.93	0.95
Adaboost	0.98	0.70	0.73	0.70

Table 1: Result of different models in terms of different metrics

The table shows the performance of diverse models by giving figures of different evaluation metrics. It's noticeable that the scores of Support Vector Machine and Random Forest were considerably high and especially, they reached the perfect state by having the accuracy score equal to 1. These models were not the only ones that had really high accuracy since the rest models, except Gaussian Mixture Model, had nearly equal correctness but disparate precision, recall, and F1-score which result in different performances.

The Support Vector Machine and Random Forest were the most efficient models since not only owned the highest accuracy score, the rest figures were still relatively large, approximately 0.9. Speaking of the recall , Support Vector Machine and Random Forest still kept their manners, on the other hand, the other score was just about 0.7. One highlight thing here is that the recall of Gaussian Naive Bayes was significantly high in comparison with its precision and F1-score while these scores were quite the same in other models. In addition, the performance of Multinomial Naive Bayes and AdaBoost were equivalent in this problem. Gaussian Mixture Model did not seem to be suitable for the solution as the result of it was dramatically low, the accuracy was about 0.6 and the other scores were only about 0.17 which were the smallest number in the table.

5 Conclusion

So far, we have solved successfully network attacks detection by using different techniques from exploratory data analysis to modeling and achieved the best result on the Random Forest model which has overall accuracy 100%, average recall 93% using the KDD Cup 99 dataset.

Although the dataset has been considered as a benchmark for network attacks detection problem for decades, it is quite outdated as it was released in the year 1999. It is also a suitable reason why we could achieve such a surprising result with some state-of-the-art techniques and models.

For the future development, we would like to use a problem-related dataset involving time series as now the network attacks are hardly spotted by not using the dependence on time.

References

[1] KDD Cup 1999 Data; 1999. Available from: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[2] Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. 2009:1-6.

[3] The Importance of Data Preparation for Business Analytics; 2019. Available from: https://www.ironsidegroup.com/2019/07/16/data-preparation-business-analytics/.

[4] Tour of Evaluation Metrics for Imbalanced Classification; 2020. Available from: https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/.

[5] Failure of Classification Accuracy for Imbalanced Class Distributions; 2020. Available from: https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/.

[6] Scikit-learn;. Available from: https://scikit-learn.org/stable/.