Network Traffic Attack Classification

Li Xiaodan
1024041114
Nanjing University of Posts and Telecommunications
School of Computer Science
Nanjing, China

Abstract— **With the rapid development and widespread application of Internet technology, the network has become an indispensable part of modern society, supporting key activities from personal communication to enterprise operation and government services. However, as the network scale expands and complexity increases, the threat of network security also becomes more serious. Network traffic attacks, as a specific type of network attack, exploit vulnerabilities in network protocols and technical implementations, manipulating or disrupting network data flows to achieve illegal purposes. These attacks can target various aspects of the network infrastructure, including routers, switches, and servers, as well as the applications that run on them, such as web services, email systems, and database management systems.This paper focuses on Distributed Denial of Service (DDoS) attacks, which have emerged as one of the most prevalent and damaging forms of network traffic attacks. DDoS attacks involve multiple compromised systems, often referred to as a "botnet," working together to flood the target with a massive volume of traffic, overwhelming its capacity to respond to legitimate requests. The impact of such attacks can be devastating, leading to significant financial losses, damage to brand reputation, and disruption of critical services.**

Index Terms— **Network Security, Traffic Attacks, Classification Methods, Application Scenarios**

I. INTRODUCTION

Network traffic attacks refer to malicious actors manipulating or disrupting data transmission on the network to achieve illegal purposes. These attacks can target network infrastructure, applications, or user data, with the aim of stealing information, disrupting services, gaining unauthorized access, or causing other forms of damage to the target system. Network traffic attacks exploit vulnerabilities in network protocols and technical implementations, often involving sending specially crafted packets, tampering with existing communication content, or abusing legitimate but vulnerable services and applications. In recent years, due to the continuous evolution of hacker tools and technologies, the economic interests behind cybercrime, the high dependence of modern society on the Internet, and the application of new technologies, network traffic attacks have become more frequent and complex.

Distributed Denial of Service (DDoS) attacks are an extended form of Denial of Service (DoS) attacks. DoS attacks are attempts to make target servers or network resources unavailable. Attackers send a large number of requests to the target, exhausting its processing capabilities or bandwidth resources, preventing legitimate users from accessing services normally. Typically, DoS attacks come from a single source, with attackers using one computer or a small number of controlled machines to send a large number of packets to the target. These requests may be legitimate service requests (such as HTTP GET requests) or specially crafted packets designed to trigger vulnerabilities or abnormal behavior in the target system. DDoS attacks use multiple geographically dispersed computers (commonly referred to as "botnets" or "zombies") to launch attacks simultaneously. Attackers first control a large number of devices (such as personal computers infected with malware, IoT devices, etc.) and then coordinate these devices to attack one or more targets simultaneously. Since the attack traffic comes from many different IP addresses, it is more difficult to track and block. Whether it is a DoS or DDoS attack, the most direct impact is the temporary or permanent unavailability of the target website, application, or service. This not only affects user experience but also leads to revenue loss, customer churn, and brand image damage. In order to improve the detection accuracy of network traffic attacks, this paper proposes an advanced classification method that integrates multiple features. This method combines machine learning technology to improve the accuracy of attack detection by analyzing multiple features in network traffic, such as traffic size, packet type, source and destination IP addresses, etc. This method can better adapt to the ever-changing network environment and can identify more complex attack patterns.

II. DATA PROCESSING

A. Data Preprocessing

The experimental design includes steps such as data preprocessing, feature selection, model training, and validation. There are two datasets for this experiment, one for training and the other for testing. Data preprocessing is a crucial part of the entire experimental process, aiming to ensure the quality and consistency of the data, providing a reliable foundation for subsequent model training and validation. First, print the dimensions and number of columns of the data, that is, the number of feature values and the amount of data, which helps us understand the basic structure and scale of the dataset. Next, filter out null values and infinite values to ensure the legality of the data during the training or validation process, avoiding model training failures or performance degradation due to data quality issues.

Since this experiment involves a labeled dataset, where "1" indicates that the traffic is a DDoS attack and "0" indicates normal traffic, the next step is to print the proportion of attack traffic and malicious traffic in both datasets. This step is crucial for assessing the balance of the dataset. If there is a significant difference in the proportion of attack traffic and normal traffic in the dataset, it may cause the model to bias towards the majority class during training, thereby affecting the model's generalization ability and prediction accuracy. Therefore, based on the proportion, it is decided whether to perform oversampling or undersampling on the data to balance the class distribution in the dataset. In this experiment, both datasets have dimensions and quantities of (400,000, 61), the label ratio of the training set is (0, 1): (200,000, 200,000), and the ratio of the test set is (0, 1): (19,614, 20,384). Through this information, we can preliminarily judge the balance of the dataset and formulate corresponding data processing strategies accordingly.

Next, print the specific content of the feature values and understand their meanings. This step is crucial for feature selection because only by deeply understanding the meaning of each feature can we accurately judge its relevance to the target variable (i.e., whether the traffic is a DDoS attack). Among the many features, select the feature values related to traffic, which helps improve the training efficiency and prediction performance of the model. For example, features related to network traffic size, packet type, source and destination IP addresses, etc., are usually closely related to DDoS attack detection. By carefully selecting features, we can build a more concise, efficient, and powerful predictive model.

The above is the entire process of data preprocessing. Through this series of rigorous steps, we have laid a solid foundation for subsequent model training and validation, ensuring the reliability and validity of the experimental results.

B. *Scikit-learn*

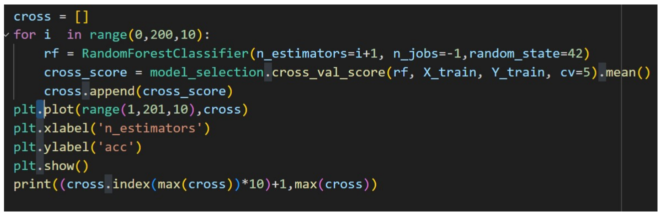
Scikit-learn (sklearn) is a commonly used third-party module in Python for machine learning, encapsulating common machine learning methods, including regression, dimensionality reduction, classification, clustering, etc. Sklearn has simple and efficient data mining and analysis tools, as well as the characteristic of being reusable in complex environments. The dataset used in this experiment is labeled, so supervised learning methods are adopted. Supervised learning is based on existing datasets where the relationship between inputs and outputs is known. Based on this known relationship, an optimal model is trained. In supervised learning, the training data includes both features and labels, and through training, the machine can find the connection between features and labels. When faced with data that only has features and no labels, it can determine the labels.

III. Experimental Process

This experiment employed six common methods from supervised learning: Classification and Regression Trees (CART), Linear Regression (LR), Naive Bayes (NB), Random Forest (RF), Linear Discriminant Analysis (LDA), and K-Nearest Neighbors (KNN). These methods range from simple linear models to complex ensemble learning models, with the aim of comparing the performance of different models on the same dataset to identify the model that best fits the current data characteristics and business needs.

During the experiment, the parameters of each model were first tuned to find the optimal values, which were then meticulously recorded. This step is crucial for optimizing model performance, as appropriate parameter settings can significantly enhance the model's predictive accuracy and generalization capability. After the tuning process, the optimal parameters of each model were compared and analyzed to assess performance differences among the models after parameter optimization, thus providing data support for the final model selection.

Taking the Random Forest (RF) model as an example, during the process of tuning its optimal parameters, we paid particular attention to the optimization of the n_estimators parameter. The parameter n_estimators represents the number of decision trees in the random forest, and its value has a significant impact on the model's performance. To find the most suitable value for n_estimators, we iterated through different values and recorded the model's performance at each setting. This process can be visually represented by plotting a line chart to illustrate the relationship between the value of the n_estimators and the performance of the model, which helps us quickly identify the optimal parameter. The following is the main code snippet for tuning the n_estimators parameter of the RF model:



```
cross = []
for i  in range(0,200,10):
    rf = RandomForestClassifier(n_estimators=i+1, n_jobs=-1,random_state=42)
    cross_score = model_selection.cross_val_score(rf, X_train, Y_train, cv=5).mean()
    cross.append(cross_score)
plt.plot(range(1,201,10),cross)
plt.xlabel('n_estimators')
plt.ylabel('acc')
plt.show()
print((cross.index(max(cross))*10)+1,max(cross))
```

Fig. 1.   RF model main codel

This code snippet demonstrates how to systematically explore the effect of varying the n_estimators parameter on the performance of a Random Forest model, providing a clear visual indication of the optimal number of trees for the ensemble.

Through the aforementioned code, we can clearly observe how the accuracy of the Random Forest model changes with the increase in the value of n_estimators. This not only helps us select the optimal value for n_estimators but also provides an intuitive understanding of the model's performance under different parameter settings. Ultimately, by comparing the optimal parameters of other models, we can determine the model that best suits the current experimental data and objectives, laying a solid foundation for subsequent model applications and further research. The results are as follows:

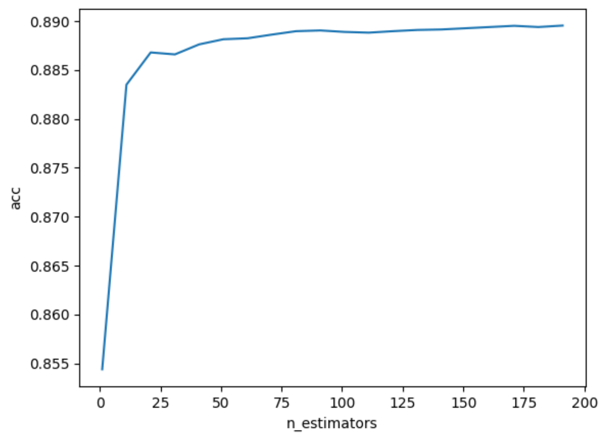


Fig. 2. n_estimators Line chart1

As can be seen from the graph, the accuracy increases with the rise in the value of n_estimators. This indicates that increasing the number of decision trees can enhance the model's performance until an optimal point is reached. Generally, increasing the value of n_estimators will add to the model's complexity and training time, but it may also improve the model's performance until an optimal point is achieved. Therefore, we continued to increase the value of n_estimators to between 200 and 350, and conducted further tuning to seek even better model performance. The results are as follows:

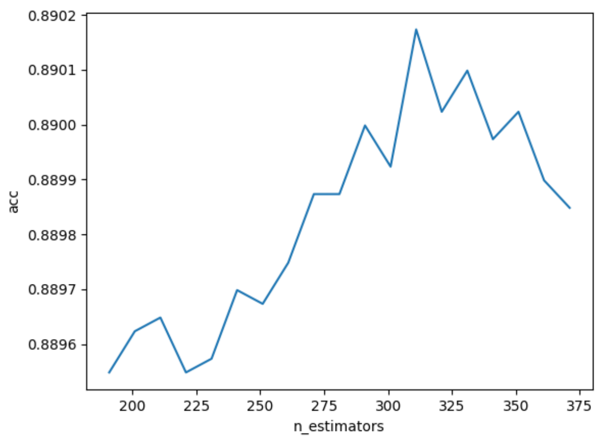


Fig. 3. n_estimators Line chart2

Through this tuning process, we found that although an increase in the value of n_estimators led to a higher accuracy, the improvement was relatively small. For instance, when n_estimator was set to 310, the accuracy reached its peak, but compared to when n_estimators was 25, the accuracy only increased by 0.005. This indicates that after reaching a certain number of decision trees, the performance improvement of the model begins to level off. At the same time, we also need to consider the computational power of the computer and the running time of the model. A higher value of n_estimators may bring a slightly higher accuracy, but it will also significantly increase the training time and consumption of computational resources.

Taking all these factors into account, we decided to set the value of n_estimators to 25. This value ensures the performance of the model while also considering computational efficiency. The optimal parameters for other models were also determined through this process, that is, by experimentation and comparison, to find the parameter settings that can achieve good performance within a reasonable computational cost.

After tuning the parameters of each model to their optimal settings, we compared the accuracy (acc) on the training set to determine which model to select for validation. The following graph shows the accuracy of each model on the training set. Through comparison, it can be seen that some models showed higher acc on the training set, but this does not necessarily mean that they will perform the same on the test set. Therefore, we comprehensively evaluated the performance of the models on the test set based on four key indicators: accuracy, precision, recall, and F1_Score. Acc reflects the proportion of correctly predicted samples out of the total samples, precision focuses on the proportion of actual positives among the samples predicted as positive, recall measures the proportion of correctly predicted positives among all actual positives, and the F1_Score, which is the harmonic mean of precision and recall, takes into account both precision and recall to provide a more comprehensive reflection of the model's performance. Acc serves as a fundamental metric for evaluating the overall correctness of a model's predictions. Calculated using the following formula: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$. In this context, TP (True Positive) represents the number of samples that the model correctly identified as positive, TN (True Negative) represents the number of samples that the model correctly identified as negative, FP(False Positive) represents the number of samples that the model incorrectly identified as positive, and FN (False Negative) represents the number of samples that the model incorrectly identified as negative. Accuracy thus reflects the proportion of correct predictions made by the model relative to the total number of samples, providing a broad assessment of the model's performance.Precision focuses on the quality of the model's positive predictions. It is defined as the proportion of actual positive samples among those predicted as positive by the model, calculated as follows: $Precision = \frac{TP}{TP+FP}$. This metric is particularly important to understand the reliability of the positive predictions of the model, to ensure that a high proportion of predicted positives are indeed true positives. Recall (also known as Sensitivity) measures the model's ability to identify all relevant positive samples. It is calculated as the proportion of actual positive samples that were correctly identified by the model: $Recall = \frac{TP}{TP+FN}$. Recall is crucial for applications where it is essential to capture as many positive instances as possible, even at the expense of some false positives. F1_Score provides a balanced measure of both precision and recall, combining these two metrics into a single value. It is defined as the harmonic mean of precision and recall, calculated as follows: $F1\_Score = 2 * \frac{Precision * Recall}{Precision + Recall}$. The F1_Score is particularly useful in scenarios where the dataset is imbalanced, as it balances the trade-off between precision and recall, offering a more comprehensive evaluation of the model's performance. Here are the training and testing results:

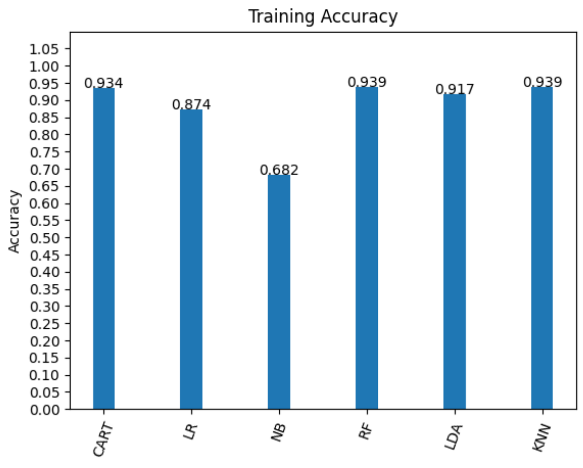


Fig. 4. Enter Caption

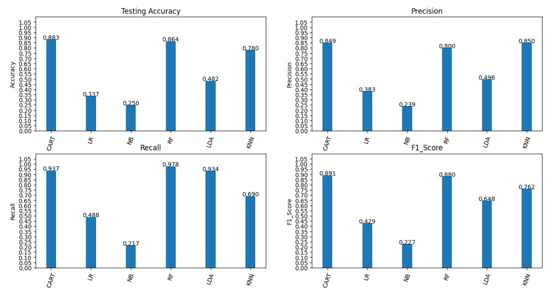


Fig. 5. Testing Results

IV. CONCLUSIONS

The experimental results indicate that CART, RF and KNN performed excellently in both training and test sets, but there was a slight decline in performance in the test set. This could be due to differences in the data distribution between the test set and the training set, or it could be caused by overfitting the models. For instance, the data in the training and validation sets may be quite similar, while the test set may contain more data that is closer to real-world usage scenarios, leading to distribution bias. Additionally, factors such as excessively high model complexity, insufficient data volume, poor data quality, or overtraining may also lead to overfitting. To enhance the generalization capability of the models and facilitate their practical application, the following measures can be taken:

1. Testing with deep learning networks: Deep learning networks can automatically learn complex features within the data and may demonstrate superior performance in handling network traffic attack detection tasks.

2. Incorporating more types of malicious attack data: By increasing the diversity and complexity of the data, the models can better adapt to various attack scenarios, thereby improving their generalization capability.

3. Applying regularization techniques: Such as L1 and L2 regularization, which involve adding penalty terms to the loss function to constrain model complexity and reduce overfitting.

4. Utilizing more advanced models: For example, models like CatBoost, which may exhibit better performance when dealing with complex datasets.

By implementing these measures, the models can be better aligned with practical application requirements, enabling faster detection of whether a network is under attack, thus preventing attacks and avoiding losses. Furthermore, visualization tools, such as grouped line charts, can be used to evaluate and compare the performance of different models on both the training and test sets, providing a more intuitive understanding of how the models' performance changes.

V. SUMMARY

The objective of this experiment was to identify a suitable model for detecting malicious network traffic. While the models performed well on the training set, the accuracy on the test set failed to reach 90%, indicating less than ideal results. This discrepancy may be attributed to differences in data distribution between the training and test sets, or it could be a result of model overfitting. For instance, some studies have shown that even web traffic detection methods with the highest accuracy of 90% on a specific dataset can experience a drop of 10% to 20% in accuracy when applied to other datasets. These methods also exhibit weak generalization capabilities in real-world scenarios, being unable to detect obfuscated malicious web traffic. Due to limitations in my computer's computational power, I was unable to train deep neural networks. However, through this experiment, I gained a thorough understanding of the principles of various models and how to tune their parameters. From the course on big data, I learned a great deal, including data visualization, data preprocessing, and how to extract useful information from data. Additionally, through related research, I became aware of advanced methods such as intelligent detection of malicious network traffic based on sample enhancement. This approach can achieve good detection results with a small number of training samples by adding a certain proportion of noise data to the training set, thereby improving the model's generalization ability and stabilizing its performance across both training and test samples. In summary, although the results of this experiment were not ideal, the knowledge and insights gained from learning and referencing other studies have provided a clearer direction for future improvements in model refinement and detection capabilities. Future work could involve extracting more effective features, experimenting with diverse noise addition methods, optimizing existing deep learning models, and increasing network depth to achieve better detection and classification outcomes. Moreover, efforts could be made to collect more complex traffic samples and conduct experiments with larger datasets to further enhance accuracy and reduce false alarm rates.

References

[1] Wang Z, Fok K W, Thing V L L. Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study[J]. Computers & Security, 2022, 113: 102542.

[2] Gao M, Ma L, Liu H, et al. Malicious network traffic detection based on deep neural networks and association analysis[J]. Sensors, 2020, 20(5): 1452.

[3] Maseer Z K, Yusof R, Bahaman N, et al. Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset[J]. IEEE access, 2021, 9: 22351-22370.