AN INTELLIGENT DDOS DETECTION SYSTEM TO RECOGNIZE AND PREVENT DDOS ATTACKS USING ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Xiangfeiyang Li[1], Jonathan Thamrun[2]

[1]Fairmont Prep Academy, 2200 W Sequoia Ave, Anaheim, CA 92801
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

In response to the increasing threat of DDoS (Distributed Denial of Service) attacks, this project investigates fortifying defenses against such malicious invasions. The project incorporates a user-friendly UI featuring two buttons: one for uploading captured traffic files and another for analysis to classify whether it's a DDoS attack. The background of the problem aspires to a robust and adaptive DDoS detection system to ensure the continuity of online services [14]. To resolve this, the project proposes an automated DDoS attack detection mechanism powered by Machine Learning and Artificial Intelligence. The application involves two pivotal experiments: the first assesses model accuracy, highlighting the Decision Tree as the most promising, while the second focuses on preventing overfitting during training, and the Random Forest Classifier stands out to this one [15]. The challenges encountered were mitigated through techniques like early stopping and regularization. The model's application across various scenarios showcased its potential for effective real-time DDoS detection.

KEYWORDS

DDoS Attack, Detection System, Artificial Intelligence, Recognization & Prevention

1. INTRODUCTION

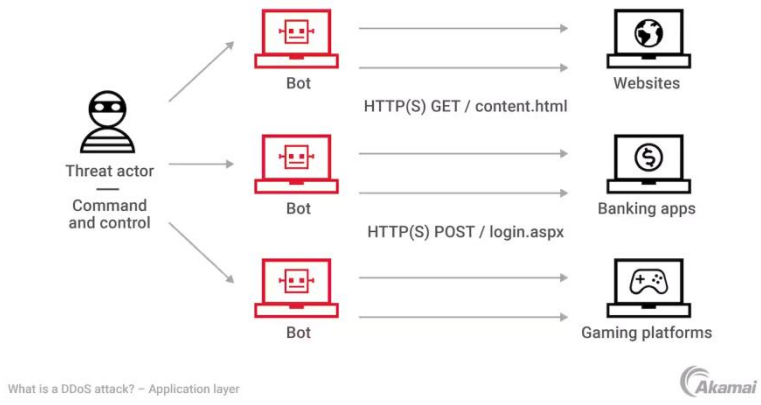

Figure 1. DDoS attack

David C. Wyld et al. (Eds): AIBD, MLSC, ACSTY, NATP, CCCIoT, SVC, SOFE, ITCSS -2024
pp. 153-167, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.140412

A DDoS(Distributed Denial of Service) attack is a malicious attempt to disturb the normal traffic of a targeted server or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic [1].
In cybersecurity, we think of the CIA triad in terms of types of attacks:

- Confidentiality: Is my information secret?
- Integrity: Is my information accurate and trustworthy?
- Availability: Can I get my information when and where I need it[2]?

Unlike other types of attacks, attackers do not use DDoS to breach your security perimeter. Instead, DDoS attacks primarily target the availability aspect of the CIA triad. DDoS attacks aim to exhaust a server or network with a massive volume of traffic, leading to a sudden surge in requests. Consequently, legitimate users are unable to access the targeted services, causing service downtime. The disruptive nature of DDoS attacks has the potential to inflict significant financial losses on businesses, especially those that heavily rely on uninterrupted online availability. Industries such as e-commerce websites and online services are particularly susceptible to these attacks, given their independence on continuous accessibility for sustaining operations. It makes them prime targets for malicious hackers seeking to exploit vulnerabilities in the digital world. As a result, organizations must invest in robust DDoS mitigation strategies to protect their online infrastructure.

In 2018, GitHub, a popular platform for software developers, experienced one of the largest DDoS attacks in history. Attackers overwhelmed the servers with a massive flood of traffic and caused intermittent outages. While GitHub quickly mitigated the attack, it disrupted the availability of its code repository services, impacting billions of developers around the world [3].

The initial methodology focuses on data preprocessing and traditional machine learning models to detect cybersecurity threats. It struggles with dimensionality reduction, therefore it potentially loses critical information. In response, my project enhances this model by incorporating a broader list of algorithms, and I focus on fine-tuning and expanding preprocessing measures as I aim for a more nuanced detection capability. The second approach leverages the flexibility of Software Defined Networking (SDN) to counteract DDoS threats. However, its reliance on conventional detection methods may lead to delays and false positives. My enhancement involves advanced hyperparameter optimization and evaluating models with extensive metrics, which aim for prompt and accurate threat detection in a dynamic network environment. This final methodology employs multiple linear regression on a benchmark dataset for threat prediction, based on their assumption of linear relationships. My project seeks to outperform this by exploring models that can analyze complex, nonlinear interactions. By doing so, it can deliver a robust and versatile detection system. Also, it transcends the linear confines of the previous approach.

By harnessing Kaggle's DDoS datasets, we could take advantage of Machine Learning to produce a comprehensive DDoS detection model and deploy it to forestall any potential DDoS threat factor. The detector is responsible for monitoring incoming network traffic and responding to that traffic with the corresponding prediction for DDoS. If a server manager is hired to organize and recognize malicious requests instead of using Artificial Intelligence, the human errors of omitting can be devastating. On the other hand, hackers in other countries use the time difference to launch DDoS attacks on servers in the early hours of the morning, when people are already resting. At this time, the server is very vulnerable due to insufficient awareness. On the contrary, Artificial Intelligence can easily address human instinctive issues.

Additionally, the second effective method to fortify network security is to activate a Web Application Firewall (WAF). A Web Application Firewall serves as a proactive defense mechanism by "sanitizing" incoming traffic and filtering out malicious requests before they even

reach the targeted server. By implementing a WAF, organizations can significantly reduce the risk of DDoS disruptions. WAF acts as a shield that deflects malicious traffic and allows traffic from legitimate users.

However, they have limitations when compared to ML methods. Firewalls operate based on predefined static rules and signatures. It makes them effective at blocking known attack patterns but less adaptive to evolving attacking threats. They also generate more false positives, block legitimate traffic, and struggle to detect complicated and dynamic DDoS attacks.

In two distinct experiments, we targeted to evaluate the accuracy metric of our 5 ML algorithms using the 20% test dataset and tried to mitigate the potential risk of overfitting. In the first experiment, we evaluated the performance of Logistic Regression, KNN, SVM, Decision Tree, and Random Forest. We trained and validated them with the same 80% of the dataset (using the same random state). The separated test dataset is used to assess their performance. Notably, the Decision Tree classifier consistently stood out with the best accuracy. Reasonably, its high accuracy could be attributed to its ability to adeptly capture complex patterns within the dataset. In the second part of the experiment, we shifted our attention to avoid overfitting, as several signs of this critical concern were displayed while evaluating the model. By implementing overfitting-killer techniques such as cross-validation, regularization, ROC AUC, and early stopping, we aimed to decrease overfitting. As a result, the Random Forest classifier, with some hyperparameter tuning, delineated the highest reduction in overfitting, ultimately enhancing its ability to generalize effectively to unseen data and showcase increased accuracy when it is implemented in real-time. These meaningful experiments portrayed the strengths and weaknesses of various models and pointed out the importance of overfitting mitigation strategies.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. The Model Architecture

AI models can be laborious to train for several reasons. One major challenge is the complexity of the model architecture, which can contain millions of parameters. As a result, they consume a vast amount of time and significant computational power to train. In addition to that, tuning these models requires meticulous multiple parameters tuning to ensure the models perform well. Confronting issues like overfitting and finding the proper balance between bias and variance can also drastically increase the training difficulty and period. Even with these efforts, the model could still possibly be onerous to converge to our expectations, leading to a series of experiments and refinement.

2.2. Dataset

Raw dataset usually contains errors and inconsistencies that are required to be identified and improved, due to the significance of ensuring high-quality data before the training processes. Subsequently, different data types such as text, number, categoricals, etc. often require different preprocessing techniques, making it challenging to create a uniform dataset. I could use Pipelines that contain a considerable amount of data preprocessor, allowing me to reduce the code complexity while still maintaining efficiency. Additionally, some datasets possess certain classes having significantly fewer samples than others. We should consider it given that imbalanced datasets can affect models' ability to predict underrepresented classes.

2.3. The Realm of Data Visualization

My third challenge involves the realm of data visualization, wherein we strive to adeptly harness the power of various types of plots. This involves discerning the unique strengths of each plot and using them to our maximum advantage. Furthermore, we're confronted with the intricate task of selecting between a bar graph and a pie graph, weighing the merits and appropriateness of each option within the context of our data representation goals. To effectively address the challenge of data visualization, I need to start by gaining a comprehensive understanding of my data and the narrative I intend to communicate. I must analyze the distinct characteristics, connections, and key takeaways from the data. Once I have established this groundwork, I can then carefully select the appropriate type of visualization.

3. SOLUTION

My main components are

1. AI components

This part consists of machine learning models that have been trained to identify DDoS attacks using patterns and features found in network traffic data. These models include Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). The detecting process is based on these core models.

2. Tkinter app

The goal of the Tkinter-based user interface (UI) is to offer a front end that is easy to use for engaging with the DDoS detection system. Users can initiate the DDoS detection procedure, upload traffic data files, and check the outcomes. The outcome is both printed in the application and saved to a text file containing details.

3. Flask web application

The system's backend, the Flask web application, manages requests from web users. This flask web application serves as an alternative to the Tkinter app. You can either upload the capture file, but also manually enter each data entry. The supplied traffic data is then analyzed, predictions are made, and the results are sent back to the front end for presentation by the Flask application using the pre-trained AI models.

The user either opens the Tkinter application or the Flask web application to upload a traffic data file or manually enters the data for the web application only. The back end then starts the DDoS detection procedure after the UI program sends a request to the bank end, acting as a bridge between the AI components and the user interface. The trained AI models are used within the application to examine the uploaded traffic statistics. Next, based on these AI models' predictions, one can determine whether the traffic data shows signs of a DDoS attack. The application receives the generated prediction results and displays them to the user. Users can take appropriate action in response to the detected threat and make well-informed decisions thanks to this intuitive interface.

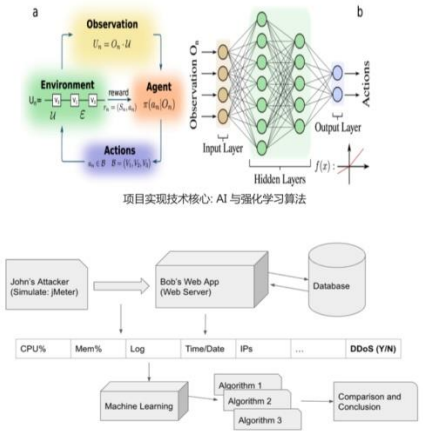

Figure 2. Overview of the solution

The key purpose of having an AI model is that it has the role of detecting abnormal traffic. We have tested out popular ML models such as KNN. KNN is an ML algorithm that assigns a label to a data point based on the majority class of its k closest (usually measured by Euclidean Distance) neighbors in the feature space, making it particularly useful for pattern recognition and similarity-based predictions. Its ability to classify instances based on their similarity to all data points makes it very suitable for a DDoS detection system. However, it follows proper hyperparameter tuning and determination of the value K for achieving highly accurate results.

```
from sklearn.neighbors import KNeighborsClassifier

neighbors = [1,3,5,10,13,15,20,25]
accuracy_knn_neighbors = []
cross_val_score_knn_neighbors = []
auc_knn_neighbors = []

for neighbor in neighbors:
    knn = KNeighborsClassifier(n_neighbors = neighbor)
    cross_val_score_knn_neighbors.append(cross_val_score(knn, X_train_ready, y_train, cv=kfold).mean()*100)
    knn.fit(X_train_ready, y_train)
    y_pred = knn.predict(X_test_ready)
    y_score = knn.predict_proba(X_test_ready)[:,1]
    accuracy_knn_neighbors.append(accuracy_score(y_test, y_pred))
    auc_knn_neighbors.append(auc(roc_curve(y_test, y_score)[0:2][0], roc_curve(y_test, y_score)[0:2][1]))


fig, (ax1, ax2, ax3) = plt.subplots(3,1,figsize=(12,36))

ax1.bar([str(n) for n in neighbors], accuracy_knn_neighbors)
ax1.set_xlabel('N')
ax1.set_ylim(min(accuracy_knn_neighbors)-0.1, max(accuracy_knn_neighbors)+0.1)
ax1.set_ylabel("Accuracy %")
ax1.set_title('KNN with different N')
for index,data in enumerate(accuracy_knn_neighbors):
    ax1.text(x=index-0.25 , y=data+(max(accuracy_knn_neighbors)-min(accuracy_knn_neighbors))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))

ax2.bar([str(n) for n in neighbors], cross_val_score_knn_neighbors)
ax2.set_xlabel('N')
ax2.set_ylim(min(cross_val_score_knn_neighbors)-0.1, max(cross_val_score_knn_neighbors)+0.1)
ax2.set_ylabel("Cross_val_score %")
ax2.set_title('KNN with different N')
for index,data in enumerate(cross_val_score_knn_neighbors):
    ax2.text(x=index-0.25 , y =data+(max(cross_val_score_knn_neighbors)-min(cross_val_score_knn_neighbors))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


ax3.bar([str(n) for n in neighbors], auc_knn_neighbors)
ax3.set_xlabel('N')
ax3.set_ylim(min(auc_knn_neighbors)-0.1, max(auc_knn_neighbors)+0.1)
ax3.set_ylabel('AUC')
ax3.set_title('KNN with different N')
for index,data in enumerate(auc_knn_neighbors):
    ax3.text(x=index-0.25 , y =data+(max(auc_knn_neighbors)-min(auc_knn_neighbors))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


plt.show()
```

```
from sklearn.svm import SVC
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

accuracy_svm_kernels = []
cross_val_score_svm_kernels = []
auc_svm_kernels = []

for kernel in kernels:
    svm = SVC(kernel=kernel, probability=True)
    cross_val_score_svm_kernels.append(cross_val_score(svm, X_train_ready, y_train, cv=kfold).mean()*100)
    svm.fit(X_train_ready, y_train)
    y_pred = svm.predict(X_test_ready)
    y_score = svm.predict_proba(X_test_ready)[:,1]
    accuracy_svm_kernels.append(accuracy_score(y_test, y_pred))
    auc_svm_kernels.append(auc(roc_curve(y_test, y_score)[0:2][0], roc_curve(y_test, y_score)[0:2][1]))

fig, (ax1, ax2, ax3) = plt.subplots(3,1,figsize=(12,36))


ax1.bar(kernels, accuracy_svm_kernels)
ax1.set_xlabel('Kernel')
ax1.set_ylim(min(accuracy_svm_kernels)-0.1, max(accuracy_svm_kernels)+0.1)
ax1.set_ylabel("Accuracy %")
ax1.set_title("SVM with different kernel")
for index,data in enumerate(accuracy_svm_kernels):
    ax1.text(x=index-0.25 , y=data+(max(accuracy_svm_kernels)-min(accuracy_svm_kernels))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


ax2.bar(kernels, cross_val_score_svm_kernels)
ax2.set_xlabel('Kernel')
ax2.set_ylim(min(cross_val_score_svm_kernels)-0.1, max(cross_val_score_svm_kernels)+0.1)
ax2.set_ylabel("Cross_val_score %")
ax2.set_title("SVM with different kernel")
for index,data in enumerate(cross_val_score_svm_kernels):
    ax2.text(x=index-0.25 , y =data+(max(cross_val_score_svm_kernels)-min(cross_val_score_svm_kernels))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


ax3.bar(kernels, auc_svm_kernels)
ax3.set_xlabel('Kernel')
ax3.set_ylim(min(auc_svm_kernels)-0.1, max(auc_svm_kernels)+0.1)
ax3.set_ylabel('AUC')
ax3.set_title("SVM with different kernel")
for index,data in enumerate(auc_svm_kernels):
    ax3.text(x=index-0.25 , y =data+(max(auc_svm_kernels)-min(auc_svm_kernels))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))

plt.show()
```

```
from sklearn.ensemble import RandomForestClassifier
n_estimators = [10,20,50,100,200,500,1000,2000]
accuracy_rfc_n = []
cross_val_score_rfc_n = []
auc_rfc_n = []
for n in n_estimators:
    forest = RandomForestClassifier(n_estimators=n)
    cross_val_score_rfc_n.append(cross_val_score(forest, X_train_ready, y_train, cv=kfold).mean()*100)

    forest.fit(X_train_ready, y_train)
    y_score = forest.predict_proba(X_test_ready)[:,1]
    y_pred = forest.predict(X_test_ready)
    accuracy_rfc_n.append(accuracy_score(y_test, y_pred)*100)
    auc_rfc_n.append(auc(roc_curve(y_test, y_score)[0:2][0], roc_curve(y_test, y_score)[0:2][1]))

fig, (ax1, ax2, ax3) = plt.subplots(3,1,figsize=(12,36))

ax1.bar([str(n) for n in n_estimators], accuracy_rfc_n)
ax1.set_xlabel("n_estimators")
ax1.set_ylim(min(accuracy_rfc_n)-0.1, max(accuracy_rfc_n)+0.1)
ax1.set_ylabel("Accuracy %")
ax1.set_title("Forest with different n_estimators")
for index,data in enumerate(accuracy_rfc_n):
    ax1.text(x=index-0.25 , y=data-(max(accuracy_rfc_n)-min(accuracy_rfc_n))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))

ax2.bar([str(n) for n in n_estimators], cross_val_score_rfc_n)
ax2.set_xlabel("n_estimators")
ax2.set_ylim(min(cross_val_score_rfc_n)-0.1, max(cross_val_score_rfc_n)+0.1)
ax2.set_ylabel("Cross_val_score %")
ax2.set_title("Forest with different n_estimators")
for index,data in enumerate(cross_val_score_rfc_n):
    ax2.text(x=index-0.25 , y =data-(max(cross_val_score_rfc_n)-min(cross_val_score_rfc_n))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


ax3.bar([str(n) for n in n_estimators], auc_rfc_n)
ax3.set_xlabel("n_estimators")
ax3.set_ylim(min(auc_rfc_n)-0.1, max(auc_rfc_n)+0.1)
ax3.set_ylabel("AUC")
ax3.set_title("Forest with different n_estimators")
for index,data in enumerate(auc_rfc_n):
    ax3.text(x=index-0.25 , y =data-(max(auc_rfc_n)-min(auc_rfc_n))/5 , s="%.3f" % data, fontdict=dict(fontsize=10))


plt.show()
```

Figure 3. Screenshot of code 1

This snippet of Python code conducts a machine learning experiment/hyperparameter tuning to determine how the number of neighbors (k) affects model performance and find the best k value featuring the kNN classifier. It does this by importing the k-Nearest Neighbors (kNN) classifier from scikit-learn. It then initializes lists of k values to hold metrics for accuracy, cross-validation scores, and Area Under the Curve (AUC). The code fits the kNN model, makes predictions on the test set, then computes the AUC based on prediction probabilities, and runs cross-validation on the training data to determine the mean accuracy for each k value. Both AUC and cross-validation scores provide huge amounts of information on whether the model is overfitting. Subplots that show these measures as bar charts are then used to illustrate the results, giving a clear visual depiction of how the model's accuracy, cross-validation score, and AUC are impacted by the choice of k. Each chart includes annotated performance metrics, enhancing interpretability. The plt.show() function is called at the end to display the generated plots. The best hyperparameter is then selected for any future kNN model. Similarly, 2 other hyperparameter tunings were done for SVC and Random Forest. A list of hyperparameters were tested and the best among them would be used.

The Tkinter app uses Python's built-in Tkinter library to generate a graphical user interface (GUI) that users can use to communicate with the DDoS detection system. This part doesn't depend on sophisticated ideas or specialized services like neural networks or natural language processing. Rather, it acts as the UI/UX frontend that makes user interactions easier. Users can upload traffic data files, initiate DDoS detection, and see the generated predictions. This UI system serves as a bridge between the user and the program's core functionality, which allows users to start and stop DDoS detection tasks without having to directly deal with the machine learning models or backend processes. It improves accessibility and usability.

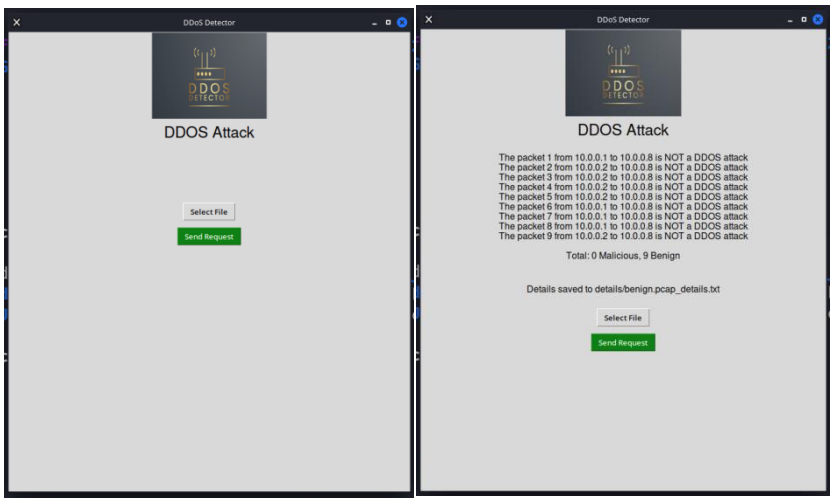

Figure 4. DDOS attack main page

```
# Create the start page
start_screen = tk.Tk()
start_screen.title("DDoS Detector")
start_screen.geometry("700x600")

# Add image to the top of the screen
start_image = Image.open("logo.png")
start_image = start_image.resize((200, 150))
start_photo = ImageTk.PhotoImage(start_image)
start_image_label = tk.Label(start_screen, image=start_photo)
start_image_label.image = start_photo
start_image_label.pack()

# Create title label
title_label = tk.Label(start_screen, text="DDOS Attack", font=("Helvetica", 20))
title_label.pack(pady=10)

# Create an indeterminate progress bar
progress_bar = ttk.Progressbar(start_screen, mode="indeterminate")
progress_bar.pack_forget()

# Create labels for analysis result and filename
result_label = tk.Label(start_screen, text="", font=("Helvetica", 12))
result_label.pack(pady=10)
filename_label = tk.Label(start_screen, text="", font=("Helvetica", 12), )
filename_label.pack(pady=10)

# Create a button to select a file
select_button = tk.Button(start_screen, text="Select File", command=select_file)
select_button.pack(pady=10)

# Create a button to start analysis
start_button = tk.Button(start_screen, text="Send Request", bg="green", fg="white", command=start_analysis)
start_button.pack()

start_screen.mainloop()
```

```
def complete_analysis(data_lists, prediction):
    global progress_bar
    global result_label
    global select_button
    global details_label
    global selected_file
    global start_button
    progress_bar.stop()  # Stop the indeterminate progress bar
    progress_bar.pack_forget()
    pred_txt = ""
    malicious = 0
    benign = 0
    for i, each in enumerate(prediction):
        pred_txt += f"The packet {i + 1} from {each[0]} to {each[1]} is {each[2]}\n"
        if each[2] == 'NOT a DDOS attack':
            benign += 1
        else:
            malicious += 1
    pred_txt += f"\nTotal: {malicious} Malicious, {benign} Benign\n"
    result_label.config(text=pred_txt)

    if not os.path.exists("details"):
        os.makedirs("details")

    with open(f"details/{os.path.split(selected_file)[1]}_details.txt", 'w') as file:
        txt = f"Total: {malicious} Malicious, {benign} Benign\n\n"
        for i, data_list in enumerate(data_lists):
            txt += f"packet {i + 1}:\n " + data_list + f"\nThe packet {i + 1} from {prediction[i][0]} to {prediction[i][1]} is {prediction[i][2]}" + "\n\n\n\n"
        file.write(txt)

    filename_label.config(text=f"Details saved to details/{os.path.split(selected_file)[1]}_details.txt")

    select_button.pack(pady=10)
    start_button.pack()
```

Figure 5. Screenshot of code 2

The Python code in question is a component of a Tkinter-built GUI application that analyzes network traffic and categorizes it as malicious or benign. After the user has uploaded the capture file, the complete_analysis function stops and hides the uploading progress bar, a UI component. It starts counters for both malicious and benign traffic, updating these counts and creating a comprehensive text summary by repeatedly running over the predictions using a loop. Furthermore, through the use of a label widget, this summary is shown in the GUI. The function then makes sure there is a directory called "details" and writes the packet's detailed summary and each of the individual predictions to a file in this directory using a naming scheme that is derived from the file that was initially chosen. In the end, the function modifies the GUI by providing the

path to the details file and adjusting the layout to possibly allow for new operations, such as selecting a new file or initiating a fresh analysis.

The Flask web application uses Bootstrap for CSS layout design and SQL for user login capabilities. Logging in securely and accessing the DDoS detection system is made possible by SQL managing user authentication. To create a front end that is both visually appealing and responsive, Bootstrap is utilized in the creation of the graphical layout and user interface. It guarantees the functionality, aesthetics, and usability of the user's interactions with the system. Finally, users can both upload traffic data files and manually enter the data to initiate DDoS detection and see the generated predictions. Similar to the Tkinter app, it improves accessibility and usability.

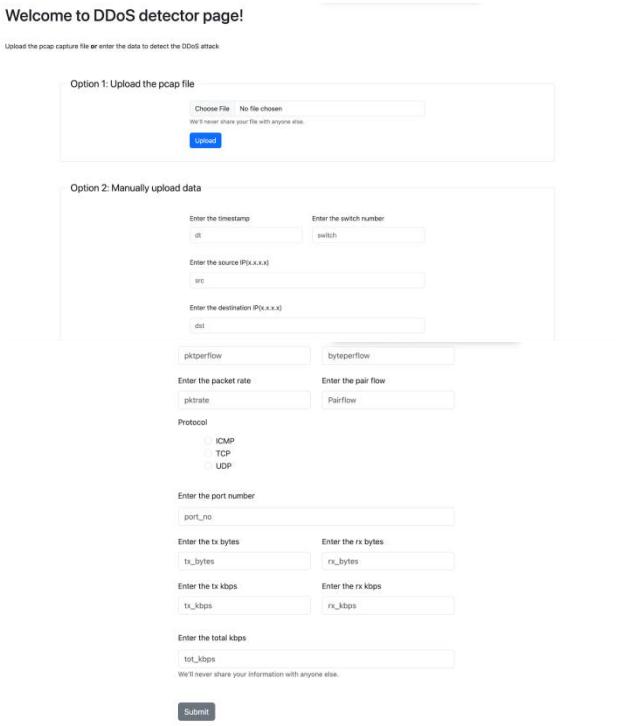

Figure 6. DDoS detector page

```
new *
@app.route( rule: '/detect_file', methods=['POST'])
def detect_file():
    if session.get('username') is not None:

        # check if the post request has the file part
        if 'file' not in request.files:
            return '<h1>No file part</h1>'
        file = request.files['file']
        # if user does not select file, browser also
        # submit an empty part without filename
        if file.filename == '':
            return '<h1>No selected file</h1>'
        if file:
            filename = file.filename
            file.save(filename)
            infos, prediction = predict(filename)

            return result_text(infos, prediction)

    else:
        return redirect(url_for('login'))
```

```python
@app.route( rule: '/detect_form', methods=['POST'])
def detect_form():
    if session.get('username') is not None:
        # get the data from the form

        dt = int(request.form['dt'])
        switch = int(request.form['switch'])
        src = request.form['src']
        dst = request.form['dst']
        pktcount = int(request.form['pktcount'])
        bytecount = int(request.form['bytecount'])
        dur = float(request.form['dur'])
        dur_nsec = int(request.form['dur_nsec'])
        tot_dur = float((dur + dur_nsec / math.pow( _x: 10,  _y: 9)) * math.pow( _x: 10,  _y: 9))
        flows = int(request.form['flows'])
        packetins = int(request.form['packetins'])
        pktperflow = int(request.form['pktperflow'])
        byteperflow = int(request.form['byteperflow'])
        pktrate = int(request.form['pktrate'])
        Pairflow = int(request.form['Pairflow'])
        Protocol = request.form['Protocol']
        port_no = int(request.form['port_no'])
        tx_bytes = int(request.form['tx_bytes'])
        rx_bytes = int(request.form['rx_bytes'])
        tx_kbps = int(request.form['tx_kbps'])
        rx_kbps = float(request.form['rx_kbps'])
        tot_kbps = float(request.form['tot_kbps'])

        data_dict = {
            'dt': dt,
            'switch': switch,
            'src': src,
            'dst': dst,
            'pktcount': pktcount,
            'bytecount': bytecount,
            'dur': dur,
            'dur_nsec': dur_nsec,
            'tot_dur': tot_dur,
            'flows': flows,
            'packetins': packetins,
            'pktperflow': pktperflow,
            'byteperflow': byteperflow,
            'pktrate': pktrate,
            'Pairflow': Pairflow,
            'Protocol_ICMP': 1 if Protocol == 'ICMP' else 0,
            'Protocol_TCP': 1 if Protocol == 'TCP' else 0,
            'Protocol_UDP': 1 if Protocol == 'UDP' else 0,
            'port_no': port_no,
            'tx_bytes': tx_bytes,
            'rx_bytes': rx_bytes,
            'tx_kbps': tx_kbps,
            'rx_kbps': rx_kbps,
            'tot_kbps': tot_kbps,
        }

        infos, prediction = predict_datalist(data_dict)

        return result_text(infos, prediction)

    else:
        return redirect(url_for('login'))
```

Figure 7. Screenshot of code 3

The code snippets are from the Flask application that shows two separate POST request processing functions, each with a different purpose. The first function, /detect_file, is dedicated to file handling; it first authenticates the user, then it searches for a file in the request and, if one is discovered, the function saves to the server temporarily and processes it with the pre-trained ML model. When the ML model has results ready, the web server presents the user with the results. The application's use in processing network traffic data manually is indicated by the second function, /detect_form, which deals with the extraction and type conversion of network-related data from a submitted form. From the screenshots of the UI, detect_form serves as an alternative to the detect_file function — it can be used if the user doesn't have a capture file but some data at hand. The form includes various metrics, like packet and byte counts and transfer rates, which are crucial for network analysis.

4. EXPERIMENT

4.1. Experiment 1

Before integrating our detector into applications, it's crucial to assess its accuracy through testing. Among the five prospective models at hand, we need to identify the optimal one. Conducting accuracy experiments, coupled with hyperparameter tuning, allows us to gauge the precision of

each model. This selection process is vital as it showcases the models' capacity to perform accurate classifications effectively.

In our meticulous evaluation process, we analyze the accuracy performance of five distinct models using the test dataset. For each model, a comprehensive set of hyperparameters is methodically fine-tuned, aiming to attain optimal accuracy outcomes. This detailed exploration of hyperparameter configurations provides valuable insights into their influence on the model's accuracy. Utilizing the carefully collected accuracy results, we generate informative visualizations that vividly illustrate the accuracy trends exhibited by each model. These visual representations serve as a powerful tool to intuitively comprehend the nuances in performance. By systematically contrasting and comparing the peak accuracy achieved by each model, we can confidently determine the most suitable model for our application's requirements. This rigorous evaluation approach ensures the selected model possesses the necessary accuracy and robustness for successful integration into real-world scenarios, where reliable performance is significant.

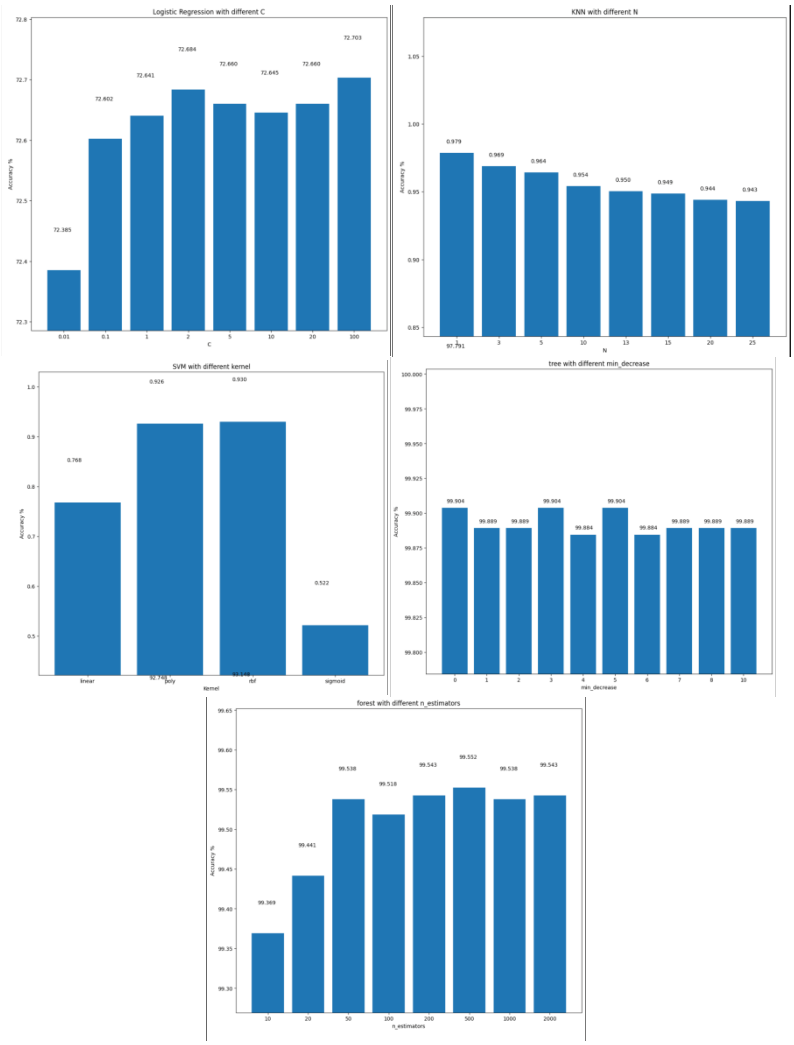

Figure 8. Figure of experiment 1

From the above visualization, we can calculate:

- Mean
  - 72.6 — Logistic Regression

- 95.7 — KNN
- 78.7 — SVM
- 99.9 — Decision Tree
- 99.5 — Random Forest

- Median
  - 72.6 — Logistic Regression
  - 95.2 — KNN
  - 84.7 — SVM
  - 99.9 — Decision Tree
  - 99.5 — Random Forest
- Min
  - 72.4 — Logistic Regression
  - 94.3 — KNN
  - 52.2 — SVM
  - 99.9 — Decision Tree
  - 99.4 — Random Forest
- Max
  - 72.7 — Logistic Regression
  - 97.9 — KNN
  - 92.0 — SVM
  - 99.9 — Decision Tree
  - 99.5 — Random Forest

Indeed, the Decision Tree model's exceptional performance, boasting an impressive accuracy of 99.9%, stands out prominently. The subsequent success of the Random Forest classifier aligns with expectations given the underlying capabilities of decision trees. Decision trees possess a distinct advantage when dealing with a mix of categorical and numerical data. Their innate ability to navigate both data types efficiently without extensive preprocessing enhances their versatility in handling diverse datasets. Moreover, decision trees are well-regarded for their proficiency in uncovering nonlinear relationships. This attribute proves invaluable when dealing with intricate interactions within the data. This capacity eliminates the necessity for convoluted mathematical transformations. However, it's vital to acknowledge that decision trees are susceptible to overfitting. Their tendency to create overly complex models that capture noise in the data can lead to decreased generalization performance. To counteract this, strategies like pruning, limiting tree depth, or employing ensemble techniques like Random Forests can help mitigate overfitting risks.

4.2. Experiment 2

Upon the conclusion of the accuracy assessment phase, it becomes evident that our models show signs of overfitting. If a model cannot generalize well to new data, then it will not be able to perform the classification or prediction tasks that it was intended for[4]. This prompts us to assess a more comprehensive evaluation to address this concern.

To achieve this, we will employ a combination of cross-validation and AUC (Area Under the Curve) analysis. By combining cross-validation and AUC, we could gain a comprehensive understanding of the model's performance in different scenarios. This assessment allows us to address the overfitting concerns and make informed decisions to ensure optimal model selection. Cross-validation provides a robust methodology to gauge the models' generalization capabilities. We could gain insights into how well the models perform across diverse data slices by partitioning the dataset into subsets. This process enables us to identify potential disparities in performance and ensures that the models can generalize effectively beyond the training data [5]. Additionally, the AUC metric plays a significant role in assessing a model's ability to

discriminate between different classes. We could gauge how well the models distinguish between positive and negative instances across various threshold levels. This analysis aids in selecting an appropriate threshold that balances precision and recall.

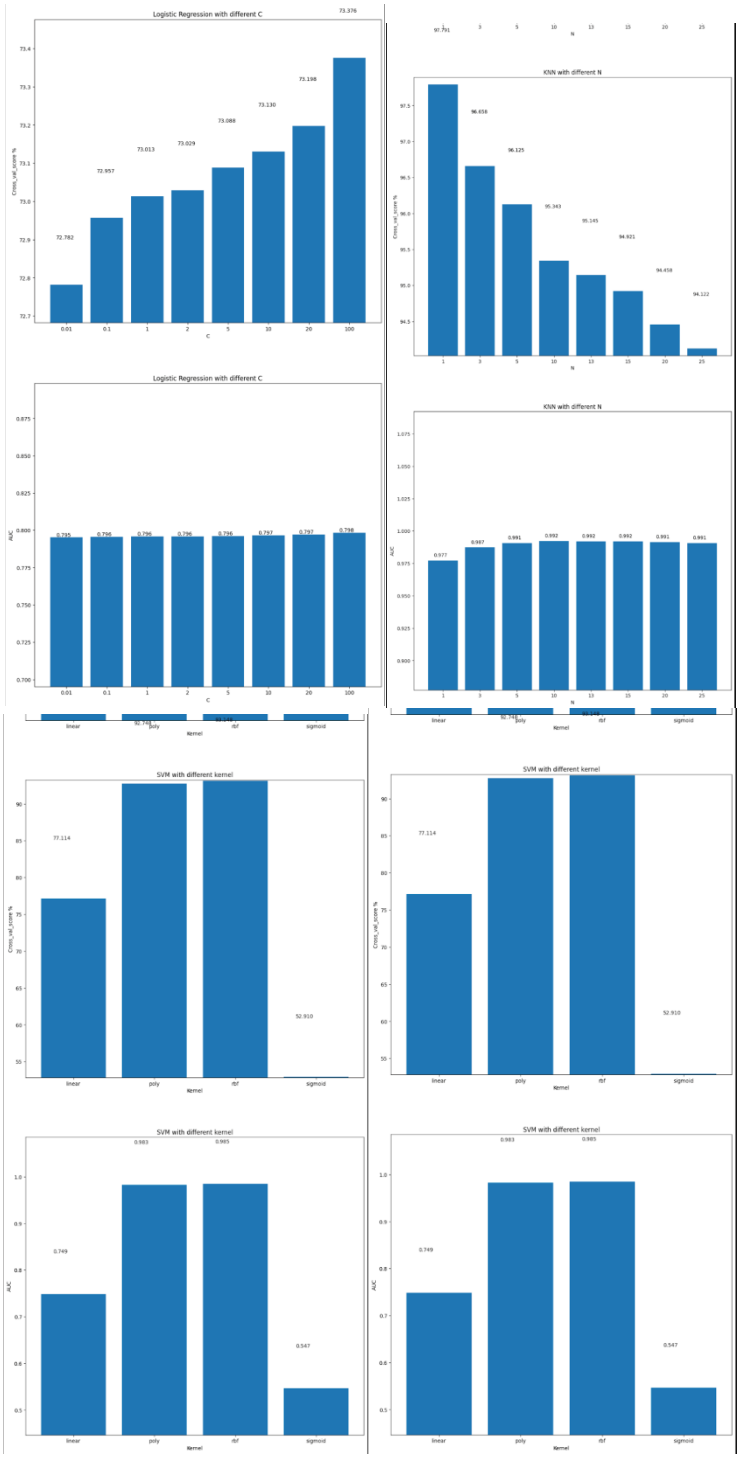

Figure 9. Figure of experiment 2

Thanks to our meticulous cross-validation and AUC assessment, we've successfully identified and excluded models exhibiting signs of severe overfitting. This step is instrumental in ensuring

that the chosen models possess the capability to perform effectively in real-world scenarios, where variations are common. A notable observation emerges from the assessments: models that display impressive accuracy during conventional evaluations experience degradation when confronted with changes. This underscores the significance of assessing model robustness beyond accuracy metrics. However, certain models such as KNN, decision trees, and random forests demonstrate excellent performance across the cross-validation and AUC evaluations. Among them, decision trees stand out, carrying the highest cross-validation accuracy, while random forests exhibit an exceptional AUC score of 1.0, despite having slightly lower cross-validation accuracy compared to decision trees. The decision to choose the random forest model as our final selection is well-founded. As an ensemble of multiple decision trees, random forests capitalize on the strength of individual trees while mitigating their weaknesses. This combination yields a model that excels in both accuracy and robustness, making it the optimal choice for deployment in real-world scenarios.

5. RELATED WORK

The proposed methodology involves 3 main components. First, data preprocessing involves handling noises, missing values, and transforming classes. The second component applies machine learning models using Scikit and Spark libraries for both regular and big data approaches. Finally, evaluation metrics are used to compare their results. Their solution's effectiveness depends on the choice of models and their hyperparameters. Limitations include potential information loss during PCA, since not all features may be equally important. The solution may not consider more advanced techniques and complex patterns. It might also overlook dynamic behaviors/changes that could affect attack detection. My project extended the scope of preprocessing and exploring a wider range of machine learning models. My project enhances the solution by incorporating additional preprocessing steps, such as deploying logistic regression, SVM, KNN, sequential neural networks, decision trees, and random forests, along with their hyperparameter tuning [6][7].

The methodology/solution proposed in the text focuses on addressing the challenges posed by Distributed Denial of Service (DDoS) attacks in the context of Software-Defined Networking (SDN). DDoS attacks are a serious threat to network security due to their potential to disrupt services and cause significant economic losses. The authors suggest leveraging the advantages of SDN, such as deep packet analysis and flexible traffic policy management, to enhance DDoS attack detection. The paper discusses various DDoS attack detection methods used in traditional network architectures, which include traffic characteristic-based detection and traffic anomaly-based detection. These methods involve creating characteristics databases, traffic modeling, and analysis of abnormal flow patterns. However, these methods can be complex, lack timely detection, or exhibit high false positive rates. The effectiveness of this solution lies in its utilization of SDN's capabilities for deep packet analysis, flexible traffic management, and quick response to policy changes. Again, my project goes beyond the focus on SVM in the research. My project also includes hyperparameter tuning for each machine learning model and comprehensive metrics evaluation.

If my project implements real-time detection, it could provide quicker responses to emerging DDoS threats [8][9][10].

The proposed methodology involves designing a machine learning model based on multiple linear regression (MLR) analysis and creating data visualization through residual plots. Their primary purpose is to apply MLR to the CICDS 2017 dataset, a benchmark dataset commonly used in research. Their process includes feature selection using the Information Gain (IG) approach. The chosen features are then subjected to MLR analysis. The effectiveness of this

solution depends on the quality/accuracy of how well these linear regression models are trained. However, this solution has its limitations. MLR assumes a linear relationship between data points. That means, if the relationship is nonlinear, the model's performance might be strongly affected. Additionally, the effectiveness of the IG relies on the assumption that the chosen features are the most relevant to our target variable. Last but not least, the solution's performance is also contingent on the quality of the CICDS 2017 dataset. On the contrary, my project investigates techniques that handle nonlinear relationships between features and the target variables. Models such as SVM (hyperplane), KNN (based on distance), and decision trees/random forests have a strong ability to distinguish complex nonlinear variable interactions [11].

6. CONCLUSIONS

One of the limitations of my project was the challenge of capturing real-life data and feeding it into our model. Real-time capture is crucial for identifying potential threats and responding promptly. I should ensure a smooth pipeline from data capture to model input to minimize the overall time. To address this limitation, I could prioritize minimizing the latency in data processing. Techniques such as stream processing could be considered to guarantee a continuous flow of real-time data to the model [12]. While minimizing latency was essential, I recognized the need to strike a balance between speed and accuracy, given there is a tradeoff. Therefore, I could tune hyperparameters based on time to ensure quick classification while still considering accuracy. This includes finding and adjusting parameters that influence both accuracy and prediction speed. Similarly, I could eliminate more features that have relatively less impact on results. In addition to traditional features, I could introduce metrics that track the frequency and nature of interactions between IP addresses and the network. These metrics provided me with a more comprehensive view of network behavior to identify subtle patterns.

In summary, this project presents a robust DDoS attack detection solution. The system achieves an accurate classification process by intricately combining efficient data preprocessing pipelines and a spectrum of machine-learning models. The project also provides two UI/UX the users can choose from, the Tkinter app and the Flask web application [13]. These 2 UI provides a friendly way of interacting with the ML model.

REFERENCES

[1] Lau, Felix, et al. "Distributed denial of service attacks." Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0. Vol. 3. IEEE, 2000.

[2] Rogers, Larry. "What Is a Distributed Denial of Service (DDoS) Attack and What Can I Do About It?." CERT Carnegie Mellon University (2004).

[3] Zaroo, Puneet. "A survey of DDoS attacks and some DDoS defense mechanisms." Advanced Information Assurance (CS 626) (2002).

[4] Ying, Xue. "An overview of overfitting and its solutions." Journal of physics: Conference series. Vol. 1168. IOP Publishing, 2019.

[5] Browne, Michael W. "Cross-validation methods." Journal of mathematical psychology 44.1 (2000): 108-132.

[6] Awan, Mazhar Javed, et al. "Real-time DDoS attack detection system using big data approach." Sustainability 13.19 (2021): 10743.

[7] Lima Filho, Francisco Sales de, et al. "Smart detection: an online approach for DoS/DDoS attack detection using machine learning." Security and Communication Networks 2019 (2019): 1-15.

[8] Ye, Jin, et al. "A DDoS attack detection method based on SVM in software defined network." Security and Communication Networks 2018 (2018).

[9] Hoque, Nazrul, Hirak Kashyap, and Dhruba Kumar Bhattacharyya. "Real-time DDoS attack detection using FPGA." Computer Communications 110 (2017): 48-58.

[10] Xu, Yang, and Yong Liu. "DDoS attack detection under SDN context." IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications. IEEE, 2016.
[11] Sambangi, Swathi, and Lakshmeeswari Gondi. "A machine learning approach for ddos (distributed denial of service) attack detection using multiple linear regression." Proceedings. Vol. 63. No. 1. MDPI, 2020.
[12] Croushore, Dean. "Frontiers of real-time data analysis." Journal of economic literature 49.1 (2011): 72-100.
[13] JOO, Heon Sik. "A Study on UI/UX and Understanding of Computer Major Students." International journal of advanced smart convergence 6.4 (2017): 26-32.
[14] Sekar, Vyas, et al. "LADS: Large-scale Automated DDoS Detection System." USENIX Annual Technical Conference, General Track. 2006.
[15] Pal, Mahesh. "Random forest classifier for remote sensing classification." International journal of remote sensing 26.1 (2005): 217-222.

© 2024 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.