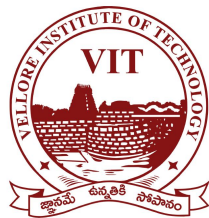


Predicting DDoS Attack using SVM

CSE 3011 Network Programming Mini Project

Prof Ganesh Reddy Karri



VIT-AP
UNIVERSITY

Submitted By:

Twinkle Kumari

20BCN7023

What is DDoS attack?

A DDoS (Distributed Denial of Service) attack is a malicious attempt to disrupt the normal functioning of a targeted server or network by overwhelming it with a flood of traffic from multiple sources. In a DDoS attack, the attacker typically uses a network of compromised computers, known as a botnet, to send a large volume of traffic to the targeted server or network, making it difficult or impossible for legitimate users to access the service.

The goal of a DDoS attack is to make a website or network unavailable by flooding it with traffic, rather than to steal data or cause damage to the system. DDoS attacks can be launched for various reasons, such as extortion, revenge, or as a form of protest. They can also be used as a distraction to divert attention away from other attacks or activities.

DDoS attacks can be difficult to defend against, as they often involve a large number of compromised computers, making it difficult to determine the source of the attack. Organizations can defend against DDoS attacks by using various techniques such as traffic filtering, load balancing, and deploying specialized DDoS mitigation tools and services.

What are the dependent factors for a successful DDoS attack?

There are several dependent factors that can contribute to the success of a DDoS attack, including:

1. **Volume of traffic:** The success of a DDoS attack depends on the amount of traffic that the attacker is able to generate. The more traffic the attacker can generate, the more likely the attack is to succeed.
2. **Number of attack sources:** DDoS attacks rely on having multiple sources of traffic, usually in the form of compromised devices or "botnets". The more sources of traffic the attacker has, the more difficult it is to defend against the attack.
3. **Attack duration:** The longer a DDoS attack lasts, the more likely it is to succeed. This is because it can take time for defenders to identify and mitigate the attack, and during that time the target may be unable to provide services to its users.
4. **Attack sophistication:** Attackers are constantly developing new techniques and tools to evade defenses and make their attacks more effective. The more sophisticated the attack, the more difficult it is to defend against.
5. **Target's infrastructure:** The success of a DDoS attack can also depend on the target's infrastructure. Some systems may be more vulnerable to certain types of attacks, and some may have limited resources for defending against DDoS attacks.

It is important to note that these factors can vary widely depending on the specific circumstances of the attack, and successful defense against DDoS attacks often requires a combination of technical and procedural measures to mitigate the impact of the attack.

What is SVM?

SVM stands for Support Vector Machine, which is a type of supervised machine learning algorithm used for classification and regression analysis.

In classification tasks, SVM attempts to find a hyperplane that can best separate two or more classes of data points. The algorithm identifies a decision boundary that maximizes the margin between the classes, meaning that it maximizes the distance between the decision boundary and the closest data points from each class. The data points closest to the decision boundary are known as support vectors, hence the name Support Vector Machine.

In regression tasks, SVM attempts to find a function that can best approximate the relationship between the input features and the target variable. The algorithm identifies a hyperplane that minimizes the error between the predicted output and the actual output.

SVM is a popular machine learning algorithm due to its effectiveness in handling high-dimensional data and its ability to generalize well to new, unseen data. SVM has been applied in various fields, such as image recognition, text classification, and bioinformatics.

Why use SVM on DDoS attack dataset?

SVM can be a useful algorithm for detecting DDoS attacks in network traffic datasets for several reasons:

1. **Non-linearity:** SVM is effective at identifying non-linear relationships between the features and the target variable. This is important because DDoS attacks can have complex patterns that may not be easily identified by linear models.
2. **High-dimensional data:** SVM can handle high-dimensional data, which is often the case with network traffic datasets. DDoS attack detection often requires analyzing a large number of features, such as packet size, source IP address, and destination port.
3. **Binary classification:** SVM is a binary classification algorithm, which means it is well-suited for identifying whether a particular network traffic pattern is malicious or

benign. This is a common goal in DDoS attack detection, where the goal is to identify traffic that is part of an attack and separate it from legitimate traffic.

4. **Robustness:** SVM is known to be robust to noise and outliers, which can be present in network traffic datasets. This means that SVM can still perform well even if the data contains some errors or anomalous patterns.

Overall, SVM is a powerful machine learning algorithm that can be effective for detecting DDoS attacks in network traffic datasets due to its ability to handle high-dimensional data, non-linearity, and robustness.

What are the disadvantages of using SVM on DDoS dataset?

While SVM is a powerful machine learning algorithm for DDoS attack detection, there are some potential disadvantages to using it on DDoS datasets:

1. **Sensitivity to parameters:** SVM performance can be sensitive to the choice of parameters, such as the kernel function and the regularization parameter. Choosing the optimal parameters can be a challenging task, especially for large and complex datasets.

2. **Computationally intensive:** SVM can be computationally intensive, especially for large and high-dimensional datasets. Training an SVM model on a large DDoS dataset can be time-consuming and resource-intensive.

3. **Imbalanced data:** DDoS attack datasets are often imbalanced, with a small number of attack instances compared to a large number of normal instances. SVM may struggle with imbalanced data, as it tends to bias towards the majority class. Special techniques, such as adjusting class weights or using cost-sensitive learning, may be needed to address this issue.

4. **Feature selection:** SVM performance can be affected by the choice of features used in the model. Selecting the most relevant features for DDoS attack detection can be challenging, and including irrelevant or redundant features can decrease the performance of the model.

5. **Generalization:** SVM performance can be affected by the generalization ability of the model. A model that is overfitting the training data may not generalize well to new, unseen data.

Overall, while SVM can be a powerful machine learning algorithm for DDoS attack detection, it is important to carefully consider the potential disadvantages and limitations of using it on DDoS datasets. Proper selection of parameters, handling imbalanced data, feature selection, and ensuring good generalization are all important considerations when using SVM for DDoS attack detection.

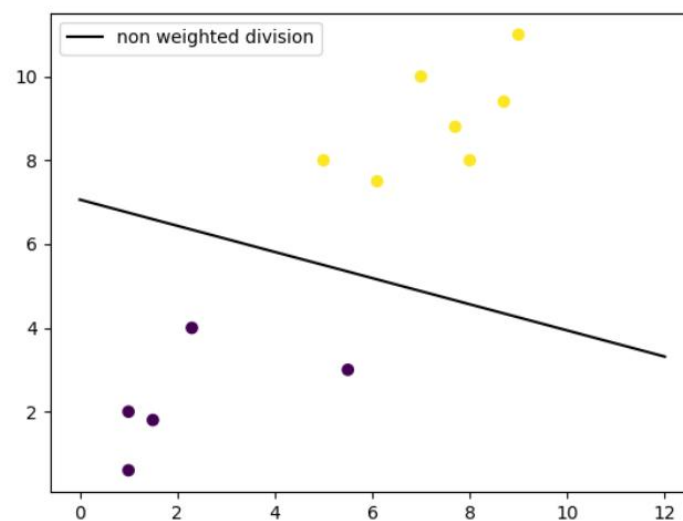
How SVM works?

A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from.

What makes the linear SVM algorithm better than some of the other algorithms, like k-nearest neighbors, is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closest data points as possible.

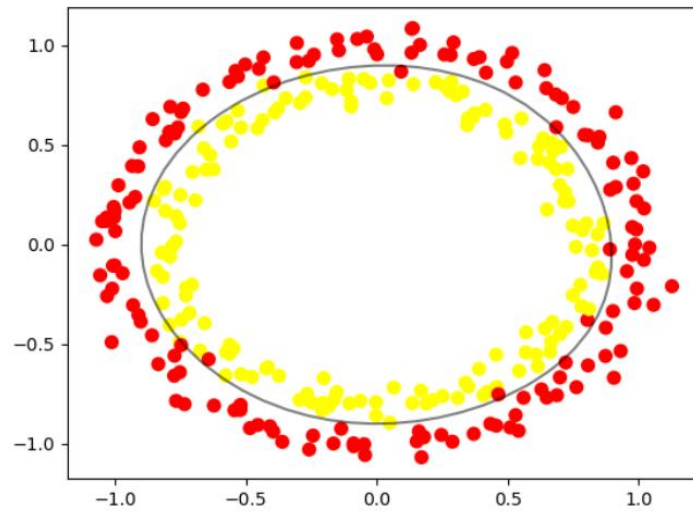
A 2-D example helps to make sense of all the machine learning jargon. Basically you have some data points on a grid. You're trying to separate these data points by the category they should fit in, but you don't want to have any data in the wrong category. That means you're trying to find the line between the two closest points that keeps the other data points separated.

So the two closest data points give you the support vectors you'll use to find that line. That line is called the decision boundary.

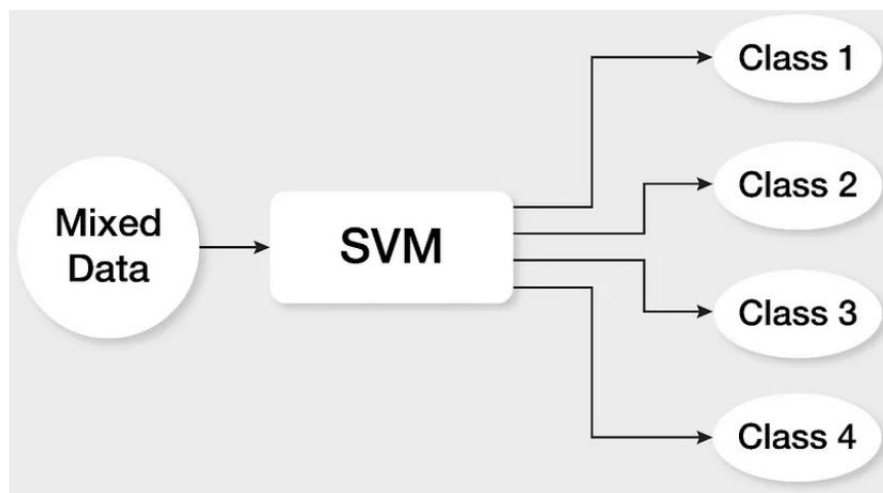


linear SVM

The decision boundary doesn't have to be a line. It's also referred to as a hyperplane because you can find the decision boundary with any number of features, not just two.



non-linear SVM using RBF kernel



+ Code + Text

Support Vector Machines

Support Vector Machine Model

```
✓ [34] from google.colab import drive  
6s drive.mount("APA-DDoS-Dataset.csv", force_remount=True)
```

Mounted at APA-DDoS-Dataset.csv

```
✓ [35] # Import libraries  
0s import pandas as pd  
import numpy as np  
  
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
✓ [36] %matplotlib inline  
0s
```

```
✓ # Import data set  
0s df=pd.read_csv("/APA-DDoS-Dataset.csv")  
df.head()
```

	ip.src	ip.dst	tcp.srcport	tcp.dstport	ip.proto	frame.len	tcp.flags.syn	tcp.flags.reset
0	192.168.1.1	192.168.23.2	2412	8000	6	54	0	0
1	192.168.1.1	192.168.23.2	2413	8000	6	54	0	0
2	192.168.1.1	192.168.23.2	2414	8000	6	54	0	0
3	192.168.1.1	192.168.23.2	2415	8000	6	54	0	0
4	192.168.1.1	192.168.23.2	2416	8000	6	54	0	0

5 rows × 9 columns

The data set is presented in a dictionary form:

```
[38] df.keys()

Index(['ip.src', 'ip.dst', 'tcp.srcport', 'tcp.dstport', 'ip.proto',
      'frame.len', 'tcp.flags.syn', 'tcp.flags.reset', 'tcp.flags.push',
      'tcp.flags.ack', 'ip.flags.mf', 'ip.flags.df', 'ip.flags.rb', 'tcp.seq',
      'tcp.ack', 'frame.time', 'Packets', 'Bytes', 'Tx Packets', 'Tx Bytes',
      'Rx Packets', 'Rx Bytes', 'Label'],
      dtype='object')
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features.

```
[40] # Create data frame
df_feat = pd.DataFrame(df[['tcp.srcport',
      'frame.len', 'Packets', 'Bytes', 'Tx Packets', 'Tx Bytes',
      'Rx Packets', 'Rx Bytes']].values,
      columns=['tcp.srcport',
      'frame.len', 'Packets', 'Bytes', 'Tx Packets', 'Tx Bytes',
      'Rx Packets', 'Rx Bytes'],
      index=df['ip.src'])
```

```
[41] df_feat.info()

<class 'pandas.core.frame.DataFrame'>
Index: 151200 entries, 192.168.1.1 to 192.168.19.1
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   tcp.srcport      151200 non-null  int64  
1   frame.len        151200 non-null  int64  
2   Packets          151200 non-null  int64  
3   Bytes            151200 non-null  int64  
4   Tx Packets       151200 non-null  int64  
5   Tx Bytes         151200 non-null  int64  
6   Rx Packets       151200 non-null  int64  
7   Rx Bytes         151200 non-null  int64  
dtypes: int64(8)
memory usage: 10.4+ MB
```

```
[42] # View data
df_feat.head()
```

	tcp.srcport	frame.len	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
ip.src								
192.168.1.1	2412	54	8	432	4	216	4	216
192.168.1.1	2413	54	10	540	5	270	5	270
192.168.1.1	2414	54	12	648	6	324	6	324
192.168.1.1	2415	54	10	540	5	270	5	270
192.168.1.1	2416	54	6	324	3	162	3	162

▼ Train Test Split

```
✓ [43] # Import function  
0s from sklearn.model_selection import train_test_split
```

```
✓ [44] # Set up x and y  
0s x = df_feat  
y = df['Packets']  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.50, random_state=50)
```

▼ Train the Support Vector Classifier

```
✓ [45] # Import model  
4m from sklearn.svm import SVC  
  
# Instantiate the model  
model = SVC()  
  
# Fit the model to the training data  
model.fit(x_train, y_train)
```

▼ SVC
SVC()

C controls the cost of missclassification on the training data.

Large value: low bias (because you penalized the cost of misclassification alot) and high variance.

***Small value:** *high bias (not penalizing the cost of missclassification as much) and low variance.

Gamma


Small: means Gaussian with a large variance

Large: high bias and low variance

▼ Predictions

```
✓ [13] # Predict using default values  
4m predictions = model.predict(x_test)
```

▼ Evaluation

```
0s  # Imports
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix
print(confusion_matrix(y_test, predictions))

# New line
print('\n')

# Classification report
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	3
2	0.00	0.00	0.00	739
3	0.00	0.00	0.00	42
4	0.00	0.00	0.00	3638
5	0.00	0.00	0.00	150
6	0.00	0.00	0.00	8248
7	0.00	0.00	0.00	299
8	0.28	1.00	0.43	10325
9	0.00	0.00	0.00	316
10	0.99	0.82	0.90	46231
11	0.00	0.00	0.00	179
12	0.00	0.00	0.00	4019
13	0.00	0.00	0.00	70
14	0.00	0.00	0.00	1083
15	0.00	0.00	0.00	19
16	0.00	0.00	0.00	216
17	0.00	0.00	0.00	1
18	0.00	0.00	0.00	21
20	0.00	0.00	0.00	1
accuracy			0.64	75600
macro avg	0.07	0.10	0.07	75600
weighted avg	0.65	0.64	0.61	75600

Code

```
# Support Vector Machines
Support Vector Machine Model
```

```
from google.colab import drive
drive.mount('APA-DDoS-Dataset.csv')
```

```
# Import libraries
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
```

```
# Import data set
df=pd.read_csv("/APA-DDoS-Dataset.csv")
df.head()
```

The data set is presented in a dictionary form:

```
df.keys()
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features.

```
df
```

```
# Create data frame
df_feat = pd.DataFrame(df[['tcp.srcport',
    'frame.len', 'Packets', 'Bytes', 'Tx Packets', 'Tx Bytes',
    'Rx Packets', 'Rx Bytes']].values,
    columns=['tcp.srcport',
    'frame.len', 'Packets', 'Bytes', 'Tx Packets', 'Tx Bytes',
    'Rx Packets', 'Rx Bytes'],
    index=df['ip.src'])
```

```
df_feat.info()
```

```
# View data
df_feat.head()
```

```
## Train Test Split
```

```
# Import function
from sklearn.model_selection import train_test_split
```

```

# Set up x and y
x = df_feat
y = df['Packets']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.50, random_state=50)

## Train the Support Vector Classifier

# Import model
from sklearn.svm import SVC

# Instantiate the model
model = SVC()

# Fit the model to the training data
model.fit(x_train,y_train)

C controls the cost of missclassification on the training data.
**Large value:** low bias (because you penalized the cost of missclassification alot)
and high variance.
**Small value:** high bias (not penalizing the cost of missclassification as much) and
low variance.

**Gamma**
**Small:** means Gaussian with a large variance
**Large:** high bias and low variance

## Predictions

# Predict using default values
predictions = model.predict(x_test)

## Evaluation

# Imports
from sklearn.metrics import confusion_matrix,classification_report

# Confusion matrix
print(confusion_matrix(y_test,predictions))

# New line
print('\n')

# Classification report
print(classification_report(y_test,predictions))

```