SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

MACHINE LEARNING BASED DETECTION OF APPLICATION LAYER DDoS ATTACK

Arun Balaji B-125157008
Jannesh Evans P-125157025
Vimal K-125157079

Guide
Dr. KANNAN BALASUBRAMANIAN, SOC

Layout of this Presentation

- Base Paper
- Motivation
- Objective
- Problem Statement
- Literature survey
- Modules
- Dataset
- Work plan
- Proposed Techniques
- Timeline For Completion
- References

Base paper

Dyari mohammed sharif,Hakem Beitollahi and mahdi fazeli, “Detection of Application-Layer DDoS Attacks Produced by Various Freely Accessible Toolkits Using Machine Learning”

Date of publication : 25 May 2023

Journal:IEEE Access

Motivation

- Distributed Denial of Service (DDoS) attacks are a growing threat to online services ,as they can overwhelm servers or networks with a flood of traffic, rendering the targeted service inaccessible to legitimate users .
- Application-layer DDoS attacks target the application layer of the victim system, aiming to exhaust its resources or cause the application to fail.
- In recent years there was a significant increase in application-layer attacks, specifically HTTP-layer DDoS attacks which rose by 164% Year over year
- The main aim here is to perform efficient prediction on the DDoS attack and the tools used to launch it.

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

Objective

- An approach to predict the application layer DDoS attacks using Machine Learning .
- To identify the tools used to launch the attack.
- To implement feature Selection techniques to enhance the speed and efficiency of the system.
- Our main aim is to fill the gap by investigating the impact of easy availability of DDoS attack tools on the frequency and severity of attacks.

Problem Statement

To construct a ML model to detect DDoS attacks by identifying and distinguishing between traffic produced by freely accessible tools and legitimate traffic.

Literature survey

Study	Main Methods used	Data	Feature Selection	Key Findings	Evaluation Metrics used	No of features used	DDoS tools addressed
[12]	SVM & DT C4.5	NSL KDD	-	DT C4.5 is found to be more accurate than SVM	Accuracy	41	No
[13]	RBF & CSA	NSL KDD	GA	The suggested found to be more accurate than other existing techniques	Sensitivity & Specificity & Positive Predictive Value & Negative Predictive Value & Precision	9	No
[14]	Rule-based classifiers	GoldenEye in CI-CID2018	Information gain, Entropy with ranker	Decision table performs better than other rule-based algorithms	accuracy & precision & true positive rate & false alarm rate	23	Yes
[15]	Filter protection strategy	LDoS attacks	-	Effort is made to motivate the development of innovative techniques to detect and fight against LDoS attacks	Positive rate & False Positive Rate	-	No
[16]	Smith-Waterman local sequence alignment	LDoS attacks	Two-threshold rule	High accuracy and F1 score are achieved in detecting LDoS attacks	Recall & Precision & Accuracy & F1 score	-	No
[17]	DT C4.5, ANN, KNN	Data collected from the switch	-	High detection rate and low false alarms are achieved	Accuracy & Sensitivity & Specificity	6	No

Literature survey

[18]	OCSA & RNN	-	OCSA	Outperforms conventional methods in precision, recall, F-measure, and accuracy	Accuracy & Precision & Recall & F1 Score	10 & 12 & 15 & 16	No
[19]	Deep Neural Network	-	Adaptive particle swarm optimization	Greater detection ratio than prior methods RAS-HO, TMS, and SVM-DoS	-	3	No
[20]	SVM & ANN & NB & DT & USML	CICID 2017	-	The USML was the most effective in accurately differentiating between Botnet and normal network traffic for Botnet DDoS attack detection, as compared to other ML algorithms	Accuracy & False Alarm Rate & Sensitivity & Specifcity & False positive rate & AUC, and Matthews correlation coefcient	9-10	No
[21]	-	-	-	Identification of IoT device flaws that make them vulnerable to DDoS attacks and the need for more intelligent defense mechanisms to fight new DDoS types and attacking methods	-	-	Yes
[22]	RF & MLP	CICID 2017	-	RF outperforms MLP	Accuracy	80	No

Modules

Module 1: Feature Selection

- Decision Trees

Module 2: Data Scaling

- Min-Max Standard Scaler

Module 3: CLASSIFIER

- Multilayer Perceptron (MLP)
- Random Search (RS)

A. DATASET

- The dataset originally has 692702 samples.
- HeartBleed attack samples are eliminated from the dataset, resulting in a final dataset of 692692 samples with having 78 features

```
# before removing heartbleed samples
data.shape
```

```
(692692, 79)
```

```
#removing heartbleed attack type since we are dealing with ddos attacks
data = data[data[' Label'] != 'Heartbleed']
```

```
# 10 heartbleed samples are removed
data.shape
```

```
(692692, 79)
```

B. DATA PREPARATION

- In this dataset, our data, with the exception of a few feature columns, was clean and ready to use
- There are a total of 1008 nan values present inside the Flow Bytes/s feature.
- In addition, there are 289 infinite values present inside the Flow Bytes/s features.
- The nan and infinite values are substituted with the median and the maximum values of the feature column.

```
#Finding Nan and inf values in Flow Bytes
nan_values = data['Flow Bytes/s'].isna().sum() #1008 nan values in flow bytes
inf_values_flow_bytes = np.isinf(data['Flow Bytes/s']).sum() # 289 inf values in flow bytes

# Calculate median and maximum values for flow bytes
median_flow_bytes = data['Flow Bytes/s'].median()
max_flow_bytes = data['Flow Bytes/s'][~np.isinf(data['Flow Bytes/s'])].max()

# Replace NaN values with median and infinite values with maximum
data['Flow Bytes/s'] = data['Flow Bytes/s'].fillna(median_flow_bytes)
data['Flow Bytes/s'] = data['Flow Bytes/s'].replace(np.inf, max_flow_bytes)
```

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

- In addition, there are 1297 infinite values present inside the Flow Packets/s features.

```
# Identify NaN and infinite values
inf_values_flow_packets = np.isinf(data[' Flow Packets/s']).sum()
max_flow_packets = data[' Flow Packets/s'][~np.isinf(data[' Flow Packets/s'])].max()

#Replace inf values with maximum values
data[' Flow Packets/s'] = data[' Flow Packets/s'].replace(np.inf, max_flow_packets)

# Confirm the changes
updated_inf_values_flow_packets = np.isinf(data[' Flow Packets/s']).sum()

print(f"Infinite values (Flow Packets/s) before: {inf_values_flow_packets}, after: {updated_inf_values_flow_packets}")
```

```
Infinite values (Flow Packets/s) before: 1297, after: 0
```

C. FEATURE SELECTION

- In our work, we use DT for the purpose of feature selection.

```
dt_classifier = DecisionTreeClassifier()
```

```
dt_classifier.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

- We populate all the data in the dataset to the DT then we consider the importance of features.

```
feature_importances = dt_classifier.feature_importances_
```

```
feature_importances
```

```
array([1.02716684e-01, 1.73139993e-04, 4.20818755e-04, 2.00379583e-03,
       0.00000000e+00, 1.08754755e-02, 5.78177418e-04, 2.43211740e-03,
       0.00000000e+00, 2.12884270e-04, 6.59430478e-05, 7.61935979e-06,
       1.57550008e-03, 4.97127589e-01, 3.01810263e-04, 4.98800178e-04,
```

- we take the mean of feature importance of all features and use it as a threshold

```
#Mean for feature importances
mean_importance = feature_importances.mean()
```

```
mean_importance
```

```
0.01282051282051282 4
```

- Any feature with feature importance below the threshold is discarded

```
selected_features = [feature for feature, importance in enumerate(feature_importances) if importance >= mean_importance]
```

```
selected_features
```

```
[0, 13, 16, 37, 40, 65]
```

- In the end of process, we are left with six features.

```python
#selected features
for i in selected_features:
    print(data.columns[i])
```

```
Destination Port             int64
Bwd Packet Length Std      float64
Flow IAT Mean              float64
Bwd Packets/s              float64
Packet Length Mean         float64
Subflow Bwd Bytes            int64
dtype: object
```

- It is worth to note that we hold all prime essential features in order to detect diverse and various DDoS attack toolkits.

D. DATA SCALING

- This work uses the Min-Max Standard Scaler to transform the data into a number between the minimum and maximum values.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaler
min_max_scaler = MinMaxScaler()
data_min_max_scaled = min_max_scaler.fit_transform(new_df)

print("Min-Max Scaled Data:")
print(data_min_max_scaled)

# Standard Scaler
standard_scaler = StandardScaler()
data_standard_scaled = standard_scaler.fit_transform(new_df)

print("\nStandard Scaled Data:")
print(data_standard_scaled)
```

E. CLASSIFIER

- Choosing the ideal number of neurons and layers in a MLP is an important step in building a neural network.
- we use MLP to classify 4 DDoS attacks and benign traffic with 6 features only
- The procedure begins with no hidden layers, and considers the accuracy, precision, recall, and F1 score as performance metrics.
- The number of hidden layers is subsequently increased incrementally, with the corresponding number of neurons determined through Gridsearch.

```python
# Define MLP classifier
mlp = MLPClassifier()
```

```python
parameter_space = {
    'hidden_layer_sizes': [(0,),(100,)],
    'activation': ['relu'],
    'solver': ['adam','lbfgs','sgd'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive'],
}
```

```python
# Perform grid search
Gridsearch = GridSearchCV(mlp, parameter_space,cv=3, verbose=10,n_jobs=-1)
Gridsearch.fit(X_train, y_train)
```

EXPERIMENTS AND RESULTS

- After feature selection, the 6 selected features are populated to a MLP of three layers.

- The first hidden layer, second hidden layer and third hidden layer consists of 100,200 and 250 neurons, respectively

- Based on the evaluation, our model obtains a level of accuracy of 98%, precision of 94%, a level of F1 score of 93%, and recall of 93% with adam optimizer.

- There are a variety of optimizers to choose when developing ML methods. SGD, LBFGS, and Adam are three famous optimizers that are frequently used for MLPs.

- After precise experimentation, we have found that Adam optimizer/solver showed the best accuracy .

```
adam_model = grid_adam.best_estimator_
```

```
y_pred_adam = adam_model.predict(X_test)
accuracy_adam = accuracy_score(y_test, y_pred_adam)
precision_adam = precision_score(y_test, y_pred_adam, average='macro')
recall_adam = recall_score(y_test, y_pred_adam, average='macro')
f1_adam = f1_score(y_test, y_pred_adam, average='macro')

print("ADAM Test Accuracy  : ", accuracy_adam*100)
print("ADAM Test Precision : ", precision_adam*100)
print("ADAM Test Recall    : ", recall_adam*100)
print("ADAM Test F1 Score  : ", f1_adam*100)
```

```
ADAM Test Accuracy  :  98.70361414475346
ADAM Test Precision :  94.1727198564275
ADAM Test Recall    :  93.02866934624976
ADAM Test F1 Score  :  93.58876645066158
```

```
sgd_model = grid_sgd.best_estimator_
```

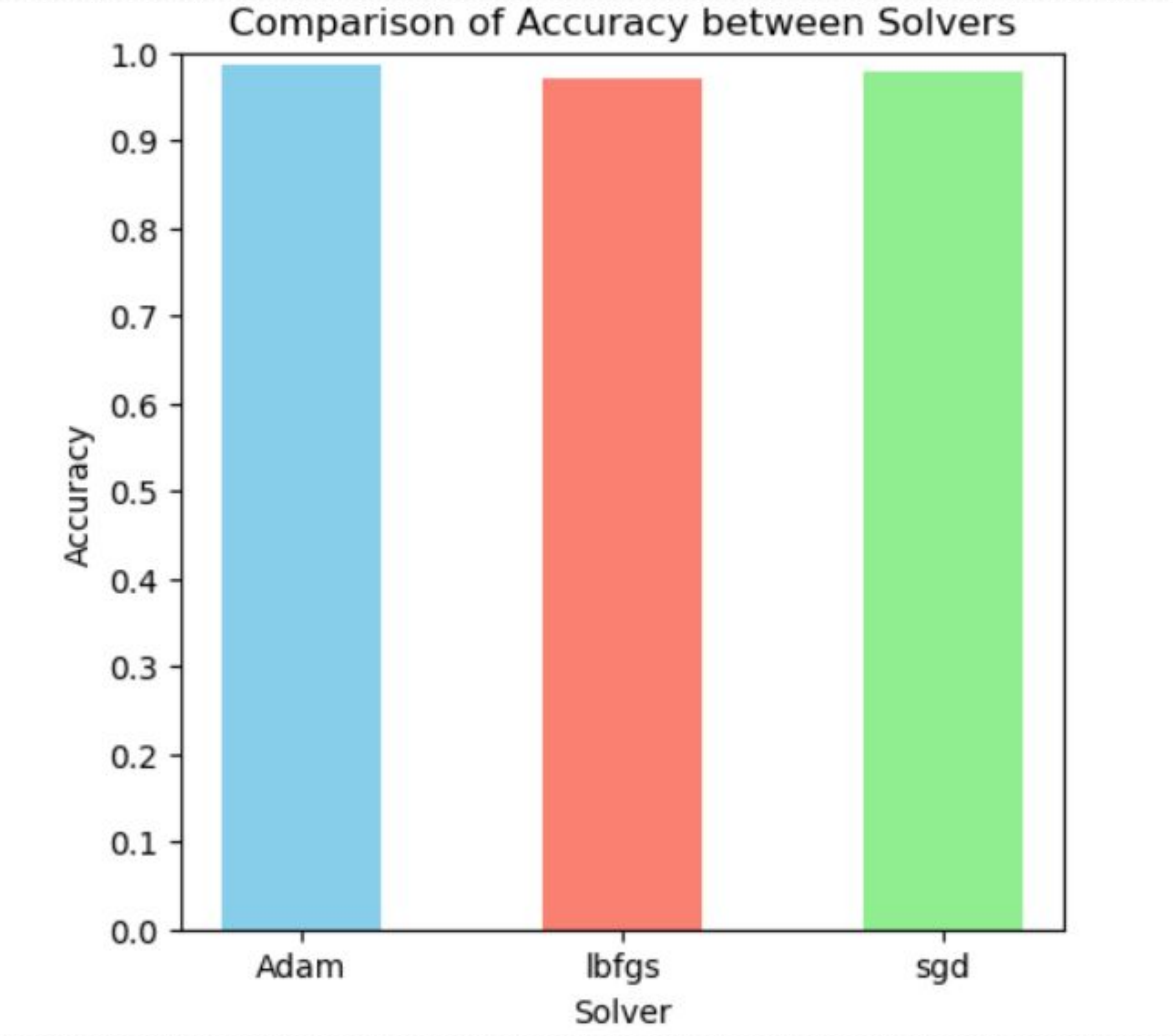
```
y_pred_sgd = sgd_model.predict(X_test)
accuracy_sgd = accuracy_score(y_test, y_pred_sgd)
precision_sgd = precision_score(y_test, y_pred_sgd, average='macro')
recall_sgd = recall_score(y_test, y_pred_sgd, average='macro')
f1_sgd= f1_score(y_test, y_pred_sgd, average='macro')

print("SGD Test Accuracy  : ", accuracy_sgd*100)
print("SGD Test Precision : ", precision_sgd*100)
print("SGD Test Recall    : ", recall_sgd*100)
print("SGD Test F1 Score  : ", f1_sgd*100)
```

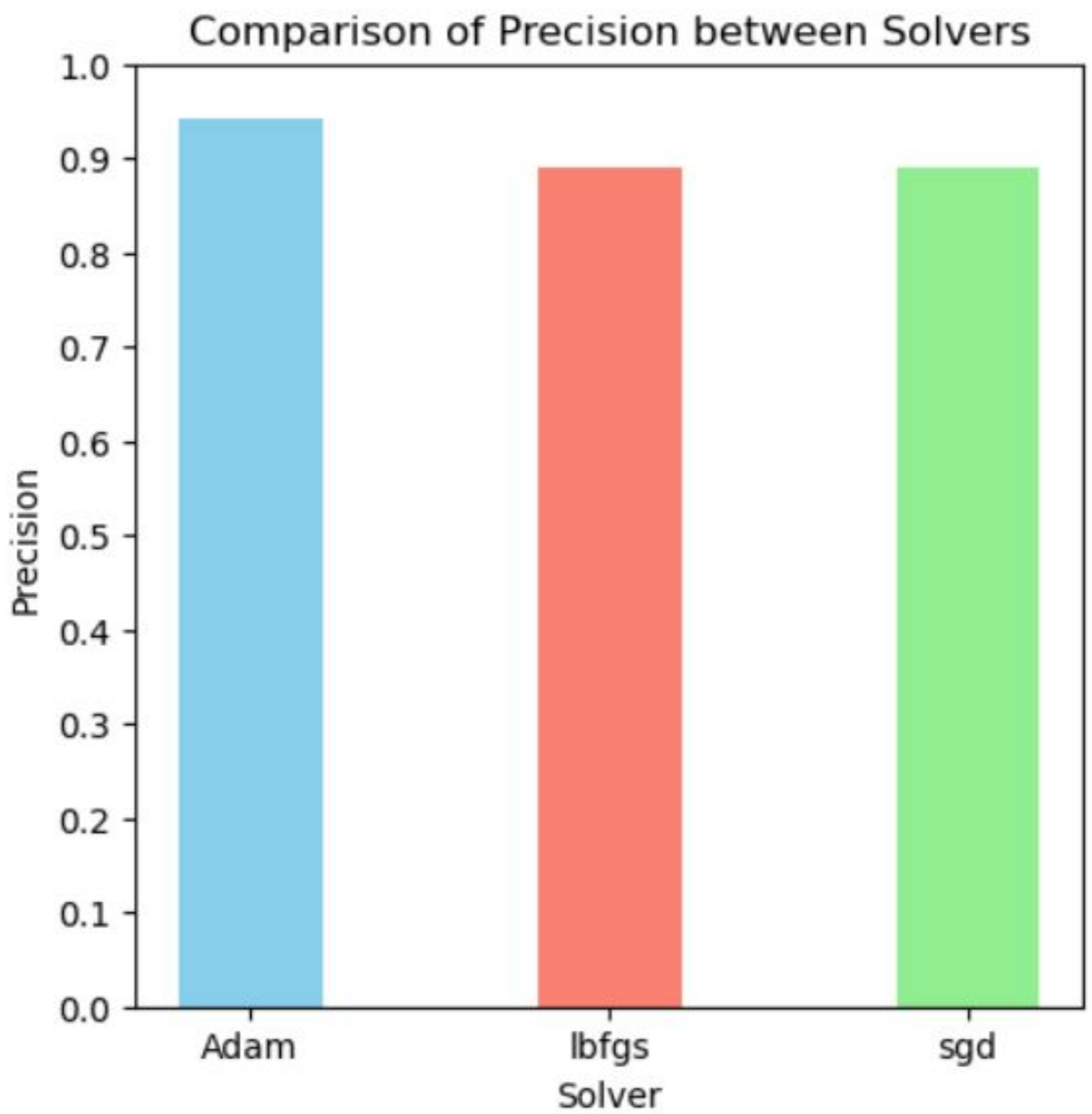
```
SGD Test Accuracy  :  97.91899753859924
SGD Test Precision :  89.0326880263286
SGD Test Recall    :  78.46240775171982
SGD Test F1 Score  :  82.91870361705875
```

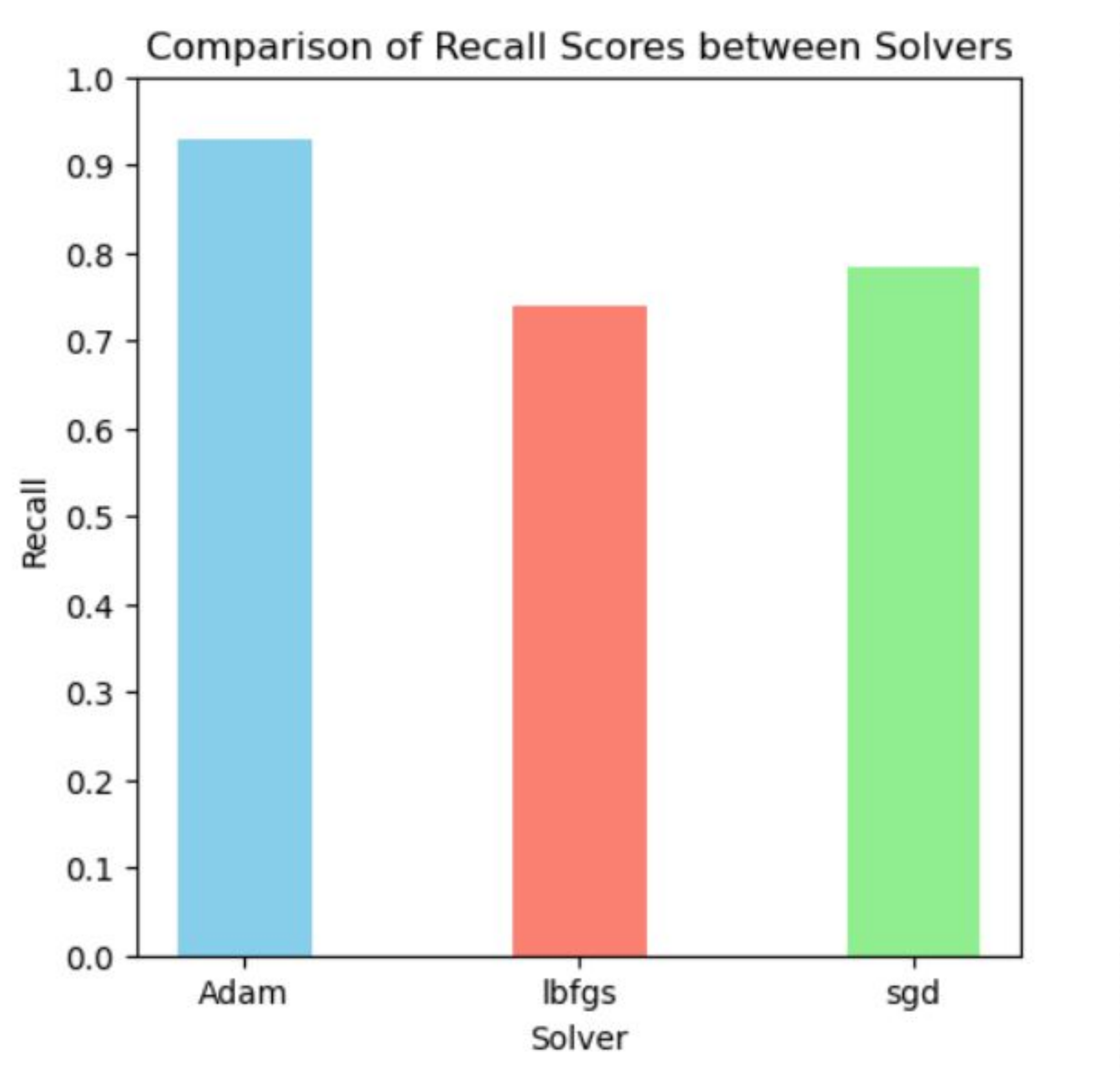
```
lbfgs_model = grid_lbfgs.best_estimator_
```

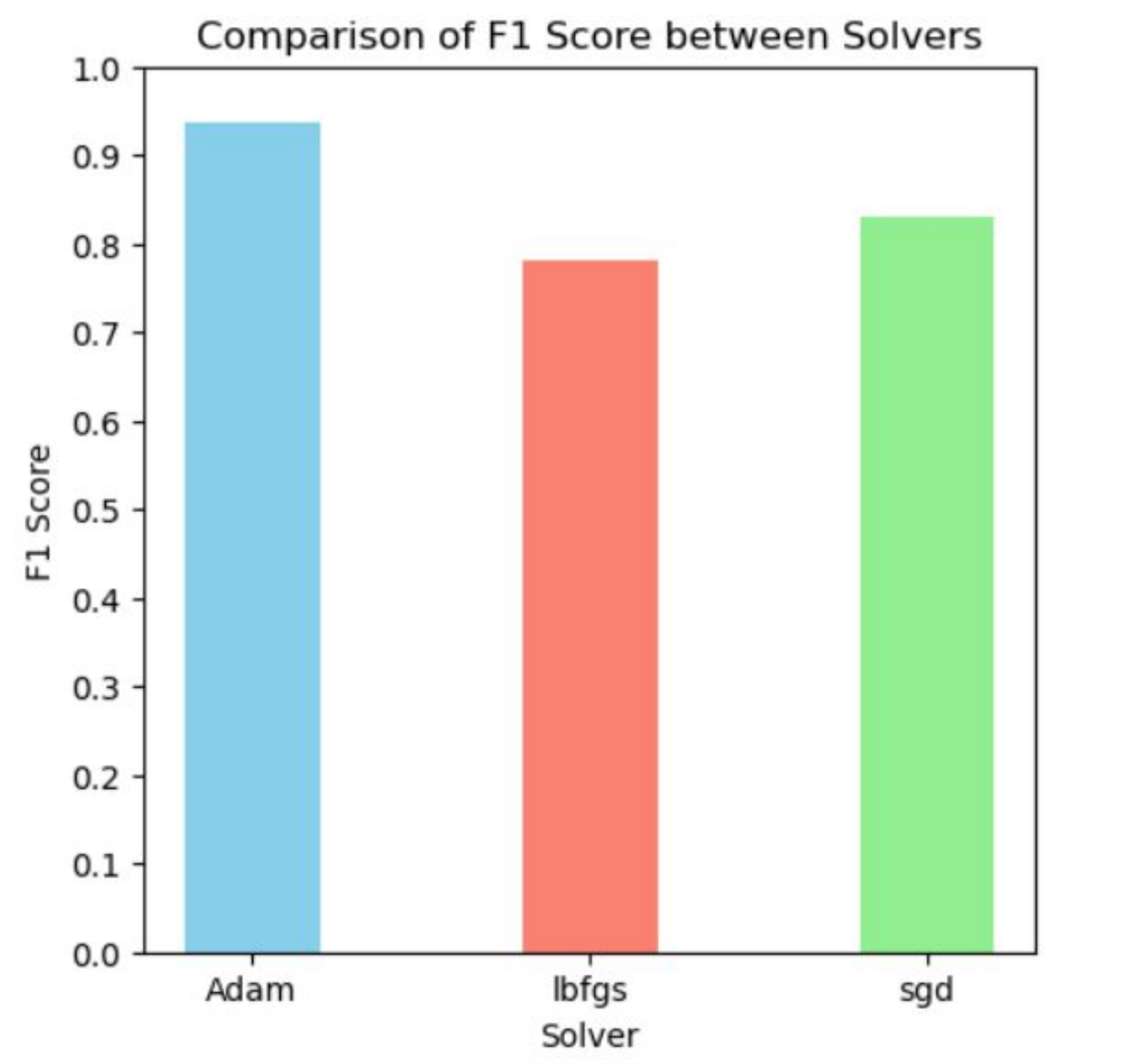
```
y_pred_lbfgs = lbfgs_model.predict(X_test)
accuracy_lbfgs = accuracy_score(y_test, y_pred_lbfgs)
precision_lbfgs = precision_score(y_test, y_pred_lbfgs, average='macro')
recall_lbfgs = recall_score(y_test, y_pred_lbfgs, average='macro')
f1_lbfgs= f1_score(y_test, y_pred_lbfgs, average='macro')

print("LBFGS Test Accuracy  : ", accuracy_lbfgs*100)
print("LBFGS Test Precision : ", precision_lbfgs*100)
print("LBFGS Test Recall    : ", recall_lbfgs*100)
print("LBFGS Test F1 Score  : ", f1_lbfgs*100)
```

```
LBFGS Test Accuracy  :  97.11344819870217
LBFGS Test Precision :  89.14146925113509
LBFGS Test Recall    :  73.92707030189567
LBFGS Test F1 Score  :  78.08444988719658
```


Comparison of Accuracy between Solvers
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Adam
lbfgs
sgd
Solver


Comparison of Precision between Solvers
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Precision
Adam
lbfgs
sgd
Solver


Comparison of Recall Scores between Solvers
Recall
Solver
Adam
lbfgs
sgd
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Comparison of F1 Score between Solvers
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
F1 Score
Adam
lbfgs
sgd
Solver

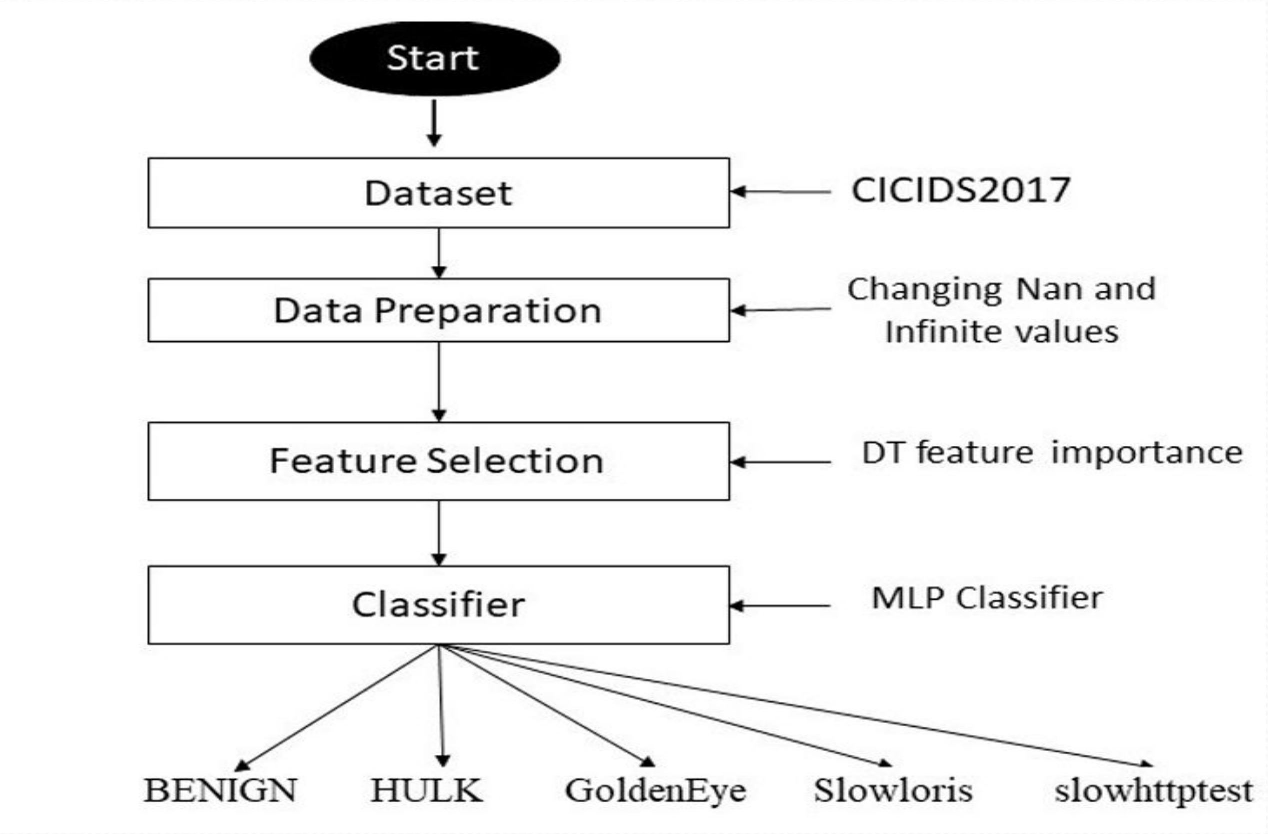
Data Set

- The CICIDS2017 dataset is used in this work where the abstract behavior of 25 users are profiled based on the HTTP, HTTPS, FTP, and SSH protocols, as well as email.
- The dataset originally has 692702 samples. HeartBleed attack samples are eliminated from the dataset, resulting in a final dataset of 692692 samples with having 78 features.
- The dataset was generated in a realistic and diverse environment, which makes it a suitable representation of real-world network traffic.

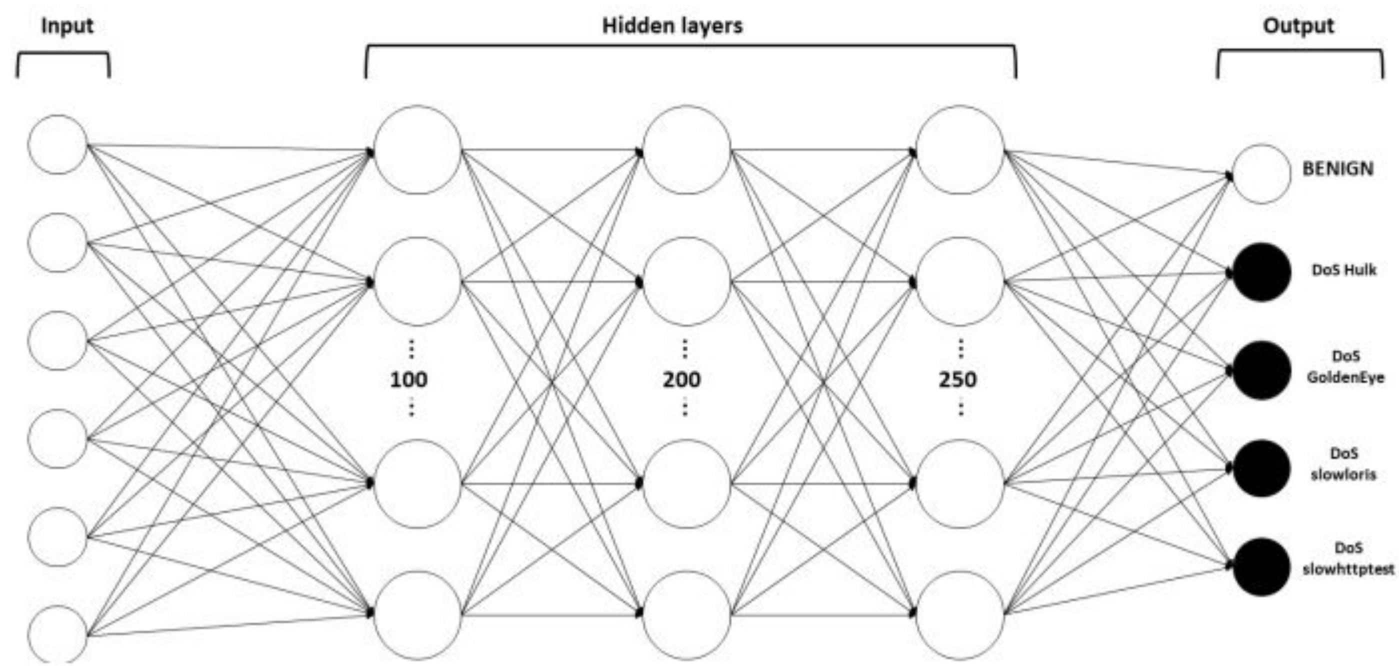
	A	B	C	D	E	F	G
1	Destination Port	Bwd Packet Length Std	Flow IAT Mean	Bwd Packets/s	Packet Length Mean	Subflow Bwd Bytes	Label
2	54865	0	3	0	6	0	BENIGN
3	55054	0	109	9174.311927	6	6	BENIGN
4	55055	0	52	19230.76923	6	6	BENIGN
5	46236	0	34	29411.76471	6	6	BENIGN
6	54863	0	3	0	6	0	BENIGN
7	54871	0	1022	0	6	0	BENIGN
8	54925	0	4	0	6	0	BENIGN
9	54925	0	42	23809.52381	6	6	BENIGN
10	9282	0	4	0	6	0	BENIGN
11	55153	0	4	0	22.66666667	0	BENIGN
12	55143	0	3	0	22.66666667	0	BENIGN
13	55144	0	1	0	22.66666667	0	BENIGN
14	55145	0	4	0	22.66666667	0	BENIGN
15	55254	0	1.5	0	12.25	0	BENIGN
16	36206	0	54	18518.51852	0	0	BENIGN
17	53524	0	1	0	0	0	BENIGN
18	53524	0	154	6493.506494	0	0	BENIGN
19	53526	0	1	0	0	0	BENIGN
20	53526	0	118	8474.576271	0	0	BENIGN
21	53527	0	239	4184.100418	0	0	BENIGN
22	53528	0	0.5	0	0	0	BENIGN
23	53527	0	1	0	0	0	BENIGN
24	55035	0	4	0	155	0	BENIGN

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

Proposed techniques (Architecture)



Proposed techniques (Architecture)



The overall structure of our proposed MLP

Timeline For Completion

- Phase 1:
  - Completion of data collection, preparation, and cleaning.
  - Implementation of feature selection and data scaling.

- Phase 2:
  - Implementation of MLP classifier with RandomizedSearchCV for model training.
  - Comparative analysis of model performance using different optimizers: Adam, SGD, and L-BFGS.

Conclusion

- In conclusion, the threat of DDoS attacks is significant due to their potential to disrupt online services and hinder user access.
- The accessibility and ease of deployment of DDoS attack tools exacerbate this concern.
- Our study employed Decision Trees for feature selection and a Multilayer Perceptron as the classifier, testing various optimizers such as Adam, SGD, and LBFGS.
- Our results consistently showed that the Adam optimizer achieved superior accuracy.

References

- B. B. Gupta, P. Chaudhary, X. Chang, and N. Nedjah, "Smart defense against distributed denial of service attack in IoT networks using supervised learning classifiers," *Comput. Electr. Eng.*, vol. 98, Mar. 2022

- H. Beitollahi and G. Deconinck, "An overlay protection layer against denial-of-service attacks"

- O. Yoachimik, "DDoS attack trends for 2022 Q1," Cloudflare, CA, USA,Tech. Rep., Apr. 2022.

- J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2,pp. 39–53, Apr. 2004

Thank you