Predicting DDoS Packet using Classification 2017310367

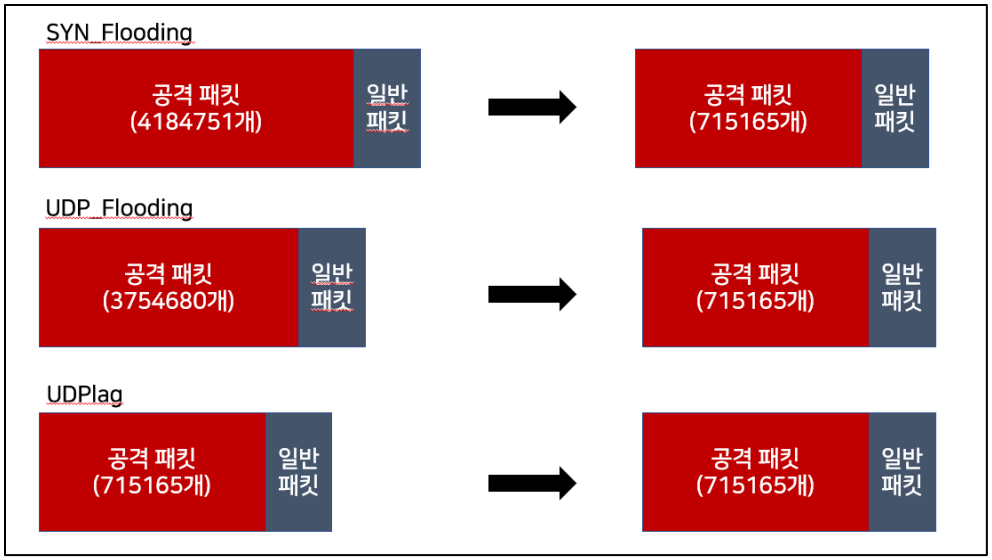
2017301367, 정다솔

Introduction

DDoS 공격이란 과도한 트래픽을 공격 대상에게 일방적으로 전송하여 서비스를 불가능하게 만드는 사이버 공격 기법이다. DoS 는 공격 PC 1 대에서 과도한 패킷을 전송하는 형태인 반면 DDoS 는 악성 코드에 감염된 여러 개의 좀비 PC 들을 이용하여 동시에 과도한 패킷을 전송하는 형태이다. 이러한 특징 때문에 DDoS 공격은 DoS 와는 비교할 수 없을 만큼 큰 트래픽이 발생하게 되며, 좀비 PC 의 개수는 DDoS 공격의 성공여부를 결정한다. 최근 들어 DDoS 공격은 IoT 기기와 결합되어 진행된다. 저장장치 용량이 제한되어 보안 시스템이 없고 24 시간 네트워크에 접속되어 있는 상태인 데다가 대부분의 IoT 기기는 단순성과 사용 편의성만을 추구하여 보안 기능이 무시되었기 때문이다. 실제로 2016 년 10 월 21 일 금요일, 전 세계 약 10 만 대의 IoT 기기에 의해, 초당 1.2 테라바이트의 데이터가 Dy 의 서버를 공격하여 미국 동부권 DNS 서버가 차단되는 사고가 발생하였으며 우리나라에서도 몇 주 전인 2021 년 4 월에 네이버가 DDoS 에 의해 공격당해 서비스가 중단되었다. 이처럼 IoT 기기의 증가로 DDoS 공격은 점점 고도화되고 있지만 이를 막을 뚜렷한 방법은 제시되지 않고 있다. IoT 기기는 특성 상 보안 업데이트가 쉽지 않아 DDoS 공격에 이용되는 것을 막기 힘들며, DDoS 공격이 진행되어도 사이버 범죄의 특성 상 공격자를 잡기 쉽지 않다. 심지어 DDoS 공격의 소스코드가 인터넷에 공개되어 있어 관심을 가진 사람이라면 누구든지 DDoS 공격을 실행할 수 있다. 그렇기에 현재로서 최선의 방법은 DDoS 공격 패킷을 미리 감지하여 이를 차단하는 것이다. 실제로 기업에서는 들어오는 패킷을 검사하는 방법을 통해 DDoS 공격을 방어하고 있다. 이에 인공지능 학습을 통해 패킷이 공격 패킷인지, 일반 패킷인지 구분하는 모델을 만들고자 한다.

Data

이번 프로젝트에 사용한 dataset 은 캐나다 사이버안전 연구소(CIC, Canadian Institute for Cybersecurity)에서 2019 년에 제공한 DDoS 공격 패킷이다. 제안서에서 발표했듯이 데이터의 크기가 너무 커 범위를 좁힐 필요가 있었다. 이에 DDoD 공격 중 악용기반공격 dataset 만을 사용하여 학습하였다. DDoS 공격의 종류는 크게 반사 기반 공격과 악용 기반 공격, 두 가지로 나뉘는데, 악용기반공격에는 SYN_Flooding, UDP_Flooding, UDPlag 공격이 있다. 이 3 가지의 공격 데이터셋이 이번 프로젝트에 사용되는 데이터이다.
그럼에도 데이터를 확인하니, 데이터의 크기가 너무 컸으며 그 중 일반 패킷은 1% 정도의 작은 비율의 차지하고 있었다. 이에 일반 패킷과 공격 패킷을 구분하여 일반 패킷은 모두 살리고, 공격 패킷의 크기는 줄였다. SYN, UDP, UDPlag 공격패킷의 비율을 맞추기 위해 min 값을 계산하여, 가장 적은 UDPlag 공격패킷에 크기에 맞추었다.



Methods

Classification 중 4 개의 class 를 각각 구분하는 multiclass classification 방법을 사용하여 학습을 진행하였다. multiclass classification 에서 주로 사용하는 MLP(multi-layer perceptron) Classifier 를 사용하였다.

Experimental Results

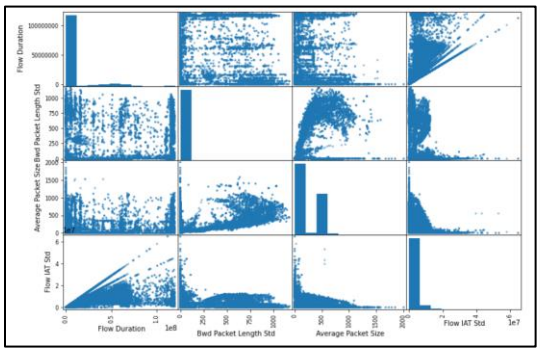
그렇게 만들어진 dataset 이 2218487 개의 column 과 89 개의 row 을 가지고 있는 것을 확인할 수 있다. 그러나 그 중 대부분이 Source IP, Destination IP, 타임 스탬프, Source 포트, Destination 포트, 프로토콜 등과 같은, DDoS 공격과는 관련 없는 항목이었다. 89 개의 feature 를 사용하면 overfitting 문제가 발생할 수 있기 때문에 feature 중에서 DDoS 패킷 탐지에 중요한 역할을 하는 feature 을 찾아내고자 하였다.

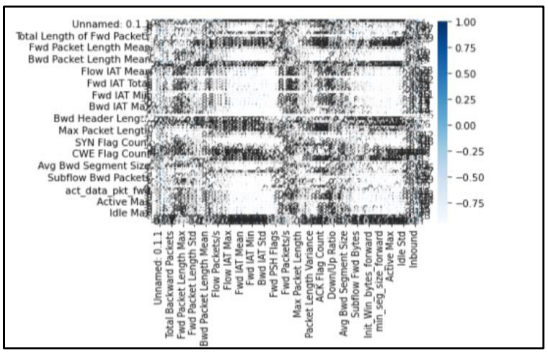
```
df.head()
```

	Unnamed: 0.1.1	Source IP	Source Port	Destination IP	Destination Port	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min
0	445444	172.16.0.5	9429	192.168.50.4	9429	6	36063894	7	2	42.0	12.0	6.0	6.0
1	113842	172.16.0.5	60224	192.168.50.4	60224	6	44851366	8	4	48.0	24.0	6.0	6.0
2	176377	192.168.50.4	11746	172.16.0.5	33827	6	1	2	0	12.0	0.0	6.0	6.0
3	24777	172.16.0.5	33828	192.168.50.4	1431	6	0	2	0	12.0	0.0	6.0	6.0
4	85100	172.16.0.5	5311	192.168.50.4	5311	6	35731470	8	2	48.0	12.0	6.0	6.0

가장 먼저, Source IP, Destination IP, Source Port, Destination Port, Protocol 과 같은 네트워크적 feature 들은 다 제외하기로 했다. 숫자형태로 되어있지 않아 계산이 어려운 데다가, DDoS 공격을 할 때는 IP 를 우회하는 것이 보통이기 때문이다. 이러한 이유로 IP, Port 와 같은 네트워크 feature 들은

DDoS 패킷의 특징을 찾아내는 데에는 필요가 없다. 또한 0 값만을 가지고 있는 feature 들도 제외했다. 학습에서 곱셈을 사용한다면, 이 feature 들에서 0 이 계속 곱해지기 때문에 학습에 도움이 되지 않는다. 그렇게 23 개의 row 를 버릴 수 있었으나 문제는 아직도 66 개의 row 가 남아있다는 것이었다. 이 문제를 해결하기 위해 DDoS 탐지 관련 논문을 찾았다. 'A Deep CNN Ensemble Framework for Efficient DDoS Attack Detection in Software Defined Networks'에 따르면 DDoS 패킷에서 가장 주요한 feature 는 Backward Packet Length (B. packet Len) Std, Flow Duration, Avg Packet Size, and Flow inter arrival time (IAT) Std 이렇게 4 개이다. 그러나 feature 가 4 개밖에 없으면 이번에는 underfitting 문제가 발생할 수 있다. 따라서 feature 들을 더 찾아내기 위해 feature 들 간의 상관관계를 분석하였다.





상관관계 분석을 통해 해당 feature 와 관련성이 높은 14 개의 feature 를 추가로 뽑아낼 수 있었다. 해당 feature 는 다음과 같으며, 최종적인 dataset 의 모습은 다음과 같다.

```
' Flow IAT Std',
' Flow IAT Mean',
' Flow IAT Min',
' Flow IAT Max',
' Fwd IAT Std',
' Fwd IAT Mean',
' Fwd IAT Max',
' Packet Length Mean',
' Packet Length Std',
' Average Packet Size',
' Min Packet Length',
' Fwd Packet Length Min',
' Fwd Packet Length Mean',
' Bwd Packet Length Std',
'Bwd Packet Length Max',
' Bwd Packet Length Mean',
' Avg Bwd Segment Size',
' Flow Duration',
'Label']]
```

Flow IAT Std	Flow IAT Mean	Flow IAT Min	Flow IAT Max	Fwd IAT Std	Fwd IAT Mean	Fwd IAT Max	Packet Length Mean	Packet Length Std	Avera Pack S
062714e+06	4.507987e+06	1.0	18628035.0	7.680923e+06	6.010649e+06	18628035.0	6.0	0.0	6.6666
056967e+06	4.077397e+06	1.0	28934293.0	1.092235e+07	6.407331e+06	28934293.0	6.0	0.0	6.5000
000000e+00	1.000000e+00	1.0	1.0	0.000000e+00	1.000000e+00	1.0	6.0	0.0	9.0000
000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.0	6.0	0.0	9.0000
026604e+06	3.970163e+06	0.0	13693985.0	6.455349e+06	5.104496e+06	13693985.0	6.0	0.0	6.6000
658033e+01	3.366667e+01	1.0	99.0	0.000000e+00	1.000000e+00	1.0	6.0	0.0	7.5000
861818e+01	3.400000e+01	1.0	52.0	0.000000e+00	4.900000e+01	49.0	6.0	0.0	7.5000
000000e+00	1.000000e+00	1.0	1.0	0.000000e+00	1.000000e+00	1.0	6.0	0.0	9.0000
081666e+00	5.033333e+01	48.0	52.0	0.000000e+00	5.100000e+01	51.0	6.0	0.0	7.5000
000000e+00	1.000000e+00	1.0	1.0	0.000000e+00	1.000000e+00	1.0	6.0	0.0	9.0000

그 후 missing value 를 mean 값을 넣어 처리하고, training set 과 test set 를 label 의 비율에 따라 split 했다. Label 의 비율을 고려하지 않으면 크기가 작은 일반 패킷이 모두 training set 으로 들어갈 수 있다.

Missing value 처리

```
[15] from sklearn.impute import SimpleImputer

     imr = SimpleImputer(missing_values=np.nan, strategy='mean')
     imr = imr.fit(ddos.values)
     imputed_data = imr.transform(ddos.values)
```

```
[16] ddos.dropna()
```

	Flow IAT Std	Flow IAT Mean	Flow IAT Min	Flow IAT Max	Fwd IAT Std	Fwd IAT Mean	Fwd IAT Max	Packet Length Mean	Pa Le
0	7.062714e+06	4.507987e+06	1.0	18628035.0	7.680923e+06	6.010649e+06	18628035.0	6.0	
1	9.056967e+06	4.077397e+06	1.0	28934293.0	1.092235e+07	6.407331e+06	28934293.0	6.0	
2	0.000000e+00	1.000000e+00	1.0	1.0	0.000000e+00	1.000000e+00	1.0	6.0	
3	0.000000e+00	0.000000e+00	0.0	0.0	0.000000e+00	0.000000e+00	0.0	6.0	
4	6.026604e+06	3.970163e+06	0.0	13693985.0	6.455349e+06	5.104496e+06	13693985.0	6.0	

Train set과 Test set split

```
[17] from sklearn.model_selection import StratifiedShuffleSplit

     split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
     for train_index, test_index in split.split(ddos, ddos["Label"]):
         ddos_train_set = ddos.loc[train_index]
         ddos_test_set = ddos.loc[test_index]
```

Training set 과 test set 을 나눈 후, training set 을 처리한다. 일단 label 부분은 4 개의 class(SYN_Flooding, UDP_Flooding, UDPlag, Benign) 로 나누어져 있기 때문에 조금 더 효율적인 학습을 위해 one-hot 인코딩을 해주었다. 처음에는 (0, 1)로 하여 공격 패킷인지 일반 패킷인지 구분하는 바이너리 classification 을 진행하고자 하였으나, 일반 패킷의 크기가 작아 이 방법은 불가능했다. 일반 패킷의 크기가 전체 패킷의 약 5% 정도를 차지하고 있어, binary classification 을 진행할 경우 모델이 모든 패킷을 공격 패킷으로 인식할 수도 있었다. 그렇게 4 개의 class 를 각각 구분하는 multiclass classification 방법을 사용하여 학습을 진행하기로 했다.

One hot 인코딩

```
[25] ddos_train_labels = pd.get_dummies(ddos_train_labels[['Label']])
```

```
[26] ddos_train_labels.head()
```

	Label_BENIGN	Label_Syn	Label_UDP	Label_UDPLag
2116146	0	1	0	0
772963	0	0	1	0
138832	0	1	0	0
1432752	0	0	1	0
1764447	0	1	0	0

Training set 의 label 을 제외한 나머지 부분은 균일화를 시켜주었다. 특정 feature 가 큰 범위의 값을 가지고 있으면 모델 전체가 그 feature 의 영향을 받을 수 있기 때문이다. Test set 도 똑같이 one-hot 인코딩과 균일화와 처리를 해준다.

StandardScaler

```
[28] from sklearn.preprocessing import StandardScaler

     stds = StandardScaler()
     ddos_train_sc = stds.fit_transform(ddos_train)
```

```
[29] ddos_train = pd.DataFrame(ddos_train_sc, index = ddos_train.index, columns=ddos_train.columns)
     ddos_train.head()
```

	Flow IAT Std	Flow IAT Mean	Flow IAT Min	Flow IAT Max	Fwd IAT Std	Fwd IAT Mean	Fwd IAT Max	Packet Length Mean	Packet Length Std	Average Packet Size	Min Packet Length	P L
2116146	-0.306231	-0.298610	-0.008099	-0.296523	-0.305430	-0.309428	-0.295700	-0.789456	-0.204042	-0.777282	-0.777553	-0.7
772963	-0.278886	-0.272302	-0.008093	-0.279570	-0.280968	-0.289112	-0.278742	1.160697	0.530260	0.984192	1.098242	1.0
138832	2.902671	2.721360	-0.008093	2.499281	2.936045	2.960004	2.500823	-0.789456	-0.204042	-0.780322	-0.777553	-0.7
1432752	-0.306249	-0.298628	-0.008093	-0.296535	-0.305430	-0.309469	-0.295711	1.325641	-0.204042	1.521351	1.405085	1.4
1764447	-0.306249	-0.298628	-0.008093	-0.296535	-0.305430	-0.309469	-0.295711	-0.789456	-0.204042	-0.771201	-0.777553	-0.7

이렇게 데이터 전처리 과정을 마친 후 학습을 시작한다. multiclass classification 에서 주로 사용하는 MLP(multi-layer perceptron)를 사용하고자 하였다.

Conclusions

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier

param_grid = [
    {'hidden_layer_sizes': [(10, 10), (15, 15), (20, 20)], 'max_iter': [20]}
  ]

MLPC = MLPClassifier(random_state=42)
grid_search = GridSearchCV(MLPC, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(ddos_train, ddos_train_labels)
```

가장 먼저, hidden layer 의 사이즈를 (10, 10), (15, 15), (20, 20)으로 하여 20 번씩 iteration 을 도는 모델을 만들었다. 정확한 결과를 관측하기 위해 grid search 를 사용하였다. 5 번씩 데이터를 학습하여 더 정확한 결과를 얻을 수 있다.

```
grid_search.best_params_
```

```
{'hidden_layer_sizes': (20, 20), 'max_iter': 20}
```

```
grid_search.best_estimator_
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(20, 20), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=20,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=42, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

그 결과, hidden layer 의 사이즈를 (20, 20)인 경우 가장 좋은 결과가 나온 것을 확인할 수 있었다.

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
0.06634599415357374 {'hidden_layer_sizes': (10, 10), 'max_iter': 20}
0.0655267106691397 {'hidden_layer_sizes': (15, 15), 'max_iter': 20}
0.06540298728978233 {'hidden_layer_sizes': (20, 20), 'max_iter': 20}
```

조금 더 자세하게 살펴보면, (10, 10)이었을 때의 mean error 은 0.06634599415357374 이며, (15, 15)일 때의 mean error 는 0.0655267106691397 이었다. 또한 (20, 20)일 때의 mean error 는 0.06540298728978233 이었다.

```
from sklearn.metrics import mean_squared_error

final_model = grid_search.best_estimator_

final_predictions = final_model.predict(ddos_test)

final_mse = mean_squared_error(ddos_test_labels, final_predictions)
final_result = np.sqrt(final_mse)
```

```
print(final_result)
```

```
0.06369754205967318
```

가장 좋은 모델이었던 (20, 20)을 가지고 test set 의 test 를 진행한 결과는 0.06369754205967318 이었다.

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print((mean_score), params)
```

```
0.9911234517245534 {'hidden_layer_sizes': (10, 10), 'max_iter': 20}
0.9912992474977728 {'hidden_layer_sizes': (15, 15), 'max_iter': 20}
0.9913043184567651 {'hidden_layer_sizes': (20, 20), 'max_iter': 20}
```

같은 조건으로 accuracy 를 구해본 결과, (10, 10)이었을 때의 accuracy 은 0.9911234517245534 이며, (15, 15)일 때의 accuracy 는 0.9912992474977728 이었다. 또한 (20, 20)일 때의 accuracy 는 0.9913043184567651 이었다. 역시 (20, 20)일 때 가장 좋은 결과를 보였다.

아쉬운 점은 조금 더 많은 모델을 실험해보고 싶었는데 시간이 부족하다는 것이었다.

Batch size 와 max_iter 도 조절하고 싶었는데 실행시간이 3 시간이 넘어가면서 런타임이 종료되고 말았다. 다음 기회가 된다면, 구분하여 모델을 만들고 싶다.

또한 생각보다 데이터가 복잡하고, 전처리를 해야 하는 부분이 많았으며 일반 패킷이 적어 학습이 어려웠다. 일반 패킷이 조금 더 풍부한 dataset 을 이용하여 학습을 했으면 더 정확한 결과를 얻을 수 있었을 것이다.

References

1. CICIDS2019, DDoS attack packet, Canadian Institute for Cybersecurity
2. 'A Deep CNN Ensemble Framework for Efficient DDoS Attack Detection in Software Defined Networks', Applied Security Engineering Research Group, Department of Computer Science, COMSATS University Islamabad, Islamabad 45550, Pakistan