Network Intrusion Detection System

4/13

1871095 Moon SooBin

1871098 Son Sumin

1971062 Kim Yujeong

Contents

- *Development Environment*
- *Data Structure*
- *Model*
- *Test*
- *Model Evaluation*

Development Environment

- ***Jupyter Notebook***
- ***Python***
- ***Tensorflow***
- ***Keras***

Data Structure

1) 데이터 전처리

데이터의 속성값

_ws.col.UTCtime, tcp.srcport, tcp.dstport, tcp.len ,tcp.seq, tcp.ack

=> 나머지 속성값들은 제외

데이터의 그룹화

```
#normal data를 그룹핑하고 100
groupset=[]
over_100_group=[]
for i in range(len(readdata.nc
    gb = readdata.normal[i].gr
    for key, group in gb:
        group = np.asarray(gro
        if len(group)>100:
            over_100_group.app
        else :
            groupset.append(group)
```

```
for i in range(len(readdata.normal)):
    gb = readdata.normal[i].groupby(['_ws.col.Protocol','ip.src','ip.dst','tcp.dstport'])
    for key, group in gb:
        group = np.asarray(group)
        if len(group)>100:
            over_100_group.append(group)
        else :
            groupset.append(group)
```

⇒ **_ws.col.Protocol, ip.src, ip.dst, tcp.srcport가 동일한 데이터끼리 그룹화**

⇒ **_ws.col.Protocol, ip.src, ip.dst, tcp.dstport가 동일한 데이터끼리 그룹화**

Data Structure

1) 데이터 전처리

데이터를 100개로 고정
100개가 넘는 데이터는 자르고, 100개가 넘지 않는 데이터는 패딩 처리하여 길이를 고정시킴

```
#normal data 100개가 넘는 그룹 100개씩
for i in range(len(over_100_group)):
    for j in range(0,len(over_100_gro
        groupset.append(over_100_grou
```

```
X=[]
for i in range(len(groupset)):
    temp=np.delete(groupset[i],[1,2,3,4,5],1)
    num=100-len(temp)
    X.append(np.pad(temp,((0,num),(0,0)),'constant', constant_values=-1))
```

Groupset에서 _ws.col.UTCtime, tcp.len ,tcp.seq, tcp.ack 속성값만 빼서 X에 넣음

```
for i in range(len(groupset)):
    groupset[i][:,0]=groupset[i][:,0]-groupset[i][0][0]
```

시간 데이터(millisecond)는 그룹별로 0부터 시작하게 만듦

Data Structure

1) 데이터 전처리

Attack data도 normal data와 똑같은 처리를 해준 후 합쳐준다.

```
X_total=np.concatenate((X_data,X_attack_data), axis=0)
```

Y data
Normal은 0, attack은 1로 세팅한다.

```
Y_data=[]
for i in range(len(X_data)):
    Y_data.append(0)
```

```
Y_a=[]
for i in range(len(X_a)):
    Y_a.append(1)
```

```
Y_total=np.concatenate((Y_data,Y_attack_data), axis=0)
```

```
X_total.shape
```

```
(104072, 100, 4)
```

```
Y_total.shape
```

```
(104072,)
```

(데이터가 다 돌아가지 않아 일부분 로드했음)

Data Structure

2) 데이터 형태

```
groups
```

```
X_a
```

```
[array
```

```
[array([[0, 14, 1, 21],
        [1, 11, 15, 55],
        [1, 6, 26, 78],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
array
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
array
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
        [-1, -1, -1, -1],
```

```
', '172.16.0.1', '192.168.10.50', 52108, 21, 14, 1, 21],
', '172.16.0.1', '192.168.10.50', 52108, 21, 11, 15, 55],
', '172.16.0.1', '192.168.10.50', 52108, 21, 6, 26, 78]],
ect),
', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 1, 21],
', '172.16.0.1', '192.168.10.50', 52112, 21, 17, 15, 55],
', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 32, 77],
', '172.16.0.1', '192.168.10.50', 52112, 21, 26, 46, 111],
', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 72, 133],
', '172.16.0.1', '192.168.10.50', 52112, 21, 20, 86, 167],
P', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 106,
dtype=object),
', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 1, 21],
', '172.16.0.1', '192.168.10.50', 52114, 21, 20, 15, 55],
', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 35, 77],
', '172.16.0.1', '192.168.10.50', 52114, 21, 17, 49, 111],
', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 66, 133],
', '172.16.0.1', '192.168.10.50', 52114, 21, 17, 80, 167],
P', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 97, 190]],
```

Model

1) Build model

학습을 진행할 모델 구현
Keras의 순차 모델을 이용해 레이어 층을 더해나간다.

```
def build_model():
    learning_rate = 0.00001
    seq_length = 100
    data_dim = 4
    METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy')
    ]
    model = Sequential()

    model.add(Conv1D(256, 3, activation = 'relu', kernel_regularizer='l2', input_shape=(100, 4)))
    model.add(MaxPooling1D(pool_size=4))
    model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
    model.add(LSTM(256))
    model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
    model.add(Dense(1, activation='sigmoid', kernel_regularizer='l2'))

    model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=learning_rate), metrics=METRICS)

    return model
```

Model

1) Build model

Model	Conv1D + LSTM (Learning rate = 0.00001)
Configuration	Conv(256) – MaxPool() – Dense() LSTM(256) – Dense() – Dense()
Compile	Binary_crossentropy

Model

1) Build model

Conv1D

Conv1D 합성곱을 통해 (100, 4) 크기의 input으로부터 특징을 추출한다.

이후, pooling 단계를 거쳐 resizing 된 새로운 레이어를 형성한다.

LSTM

훈련이 진행될수록 발생하는 장기 의존성 문제를 RNN의 LSTM 모델을 이용해 보완한다.

Entropy

결과 값이 참과 거짓으로 나오기 때문에 교차 엔트로피로 BINARY_CROSSENTROPY를 이용했다.

Model

2) Train model

```
Epoch 1/60
2917/2917 [==============================] - 47s 13ms/step - loss: 3.3866 - accura
cy: 0.8871
Epoch 2/60
2917/2917 [==============================] - 38s 13ms/step - loss: 2.0808 - accura
cy: 0.99980s - loss: 2.083
Epoch 3/60
2917/2917 [==============================] - 39s 13ms/step - loss: 1.3848 - accura
cy: 0.99980s - loss: 1.3857 - accu
Epoch 4/60
2917/2917 [==============================] - 38s 13ms/step - loss: 0.9768 - accura
cy: 0.99990s - l
```

batch_size = 128, epochs = 60 으로 설정해 전처리가 된 데이터를 훈련시킨다.

Training에 10번의 교차 검증을 진행했으며, 매 단계의 훈련이 끝날 때마다 accuracy와 f1-score을 확인했다.

```
all_history=[]
all_acc=[]
all_f1=[]
matrix=[]

model =  build_model()
for train_index, test_index in skf.split(X_total,Y_total):
    #print('train_index : ', train_index,'₩ntest_index : ',test_index)

    sm = SMOTE()
    X_train=np.reshape(X_train,(len(X_train),400))

    X_train_oversampled, y_train_oversampled = sm.fit_sample(X_train, y_train)
    X_train_oversampled=np.reshape(X_train_oversampled,(len(X_train_oversampled),100,4
    history =model.fit(X_train_oversampled, y_train_oversampled,epochs=60, batch_size=
    all_history.append(history.history)
    X_test=X_test.astype('float32')
    y_pred = model.predict(X_test)
    y_pred_binary=np.around(y_pred)

    acc=accuracy_score(y_test, y_pred_binary)
    all_acc.append(acc)

    f1=f1_score(y_test, y_pred_binary)
    all_f1.append(f1)

    matrix.append(confusion_matrix(y_test, y_pred_binary))

    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_binary).ravel()
    print(f'Accuracy: {acc}')
    print(f'f-score: {f1}')
    print(f'tn fp fn tp :{tn, fp, fn, tp}')
```

Model

3) Training result

10번의 교차 검증에서 나온 accuracy와 f1-score의 평균을 확인한 결과,
Accuracy는 0.99이며 F1-score은 0.95의 결과가 나왔음을 확인할 수 있다.

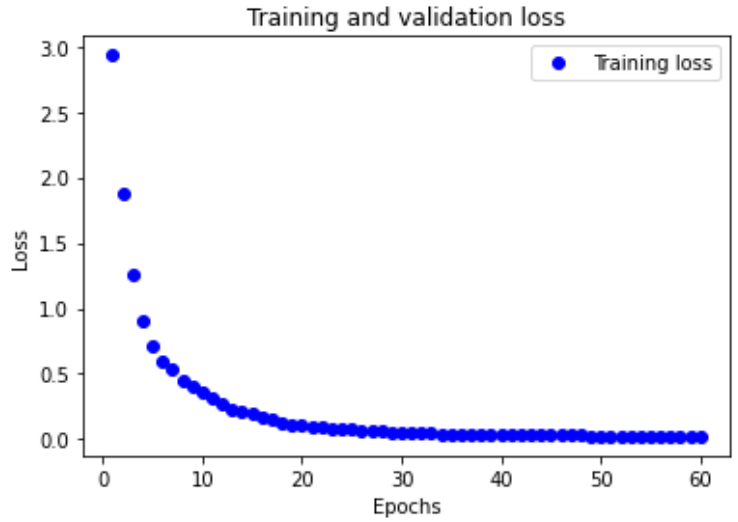
```
print(sum(all_acc)/10)
print(sum(all_f1)/10)
for i in range(k):
    print(matrix[i][0][0],matrix[i][0][1],matrix[i][1][0],matrix[i][1][1])
```

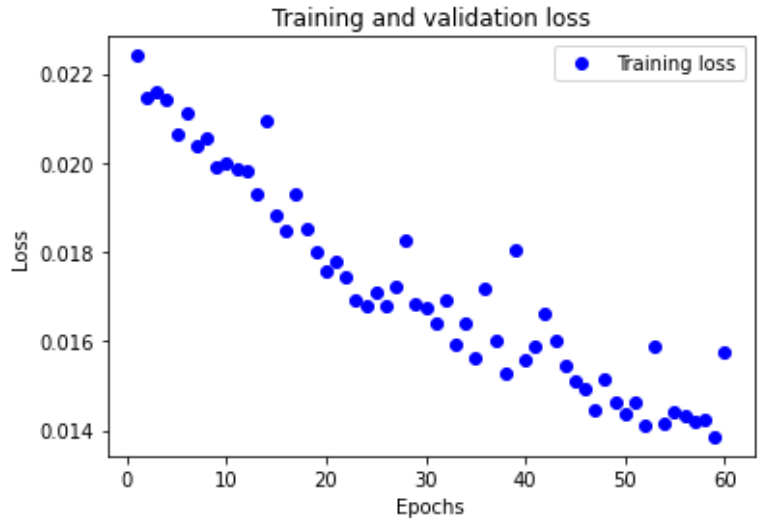
```
0.9997021521906226
0.9568071122440266
```

Model

3) Training result

```
for i in range(5):
    loss =all_history[i]['loss']
    #val_loss = all_history[i]['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```





모델 훈련을 진행하며 교차검증에서 epoch마다 나타나는 loss 값을 그래프로 표현하였다.

단계가 진행될수록 loss 값이 감소하는 것을 확인할 수 있다.

Test

모델에 테스트 데이터 적용

```
test_data=pd.read_csv('test_dataset_01.csv',error_bad_lines=False)
```

```
test_data=test_data.iloc[:,0:9]
```

```
test_data.columns=['_ws.col.UTCtime',"_ws.col.Protocol","ip.src","ip.dst","tcp.srcport","tcp.dstport","tcp.len","tcp.seq","tcp.ack"]
```

- 테스트 데이터 전처리

테스트 데이터 불러와서 속성값 설정

_ws.col.UTCtime, _ws.col.Protocol, ip.src, ip.dst, tcp.srcport, tcp.dstport, tcp.len ,tcp.seq, tcp.ack

(나머지 속성값들은 제외)

```
test_data['_ws.col.UTCtime']=test_data['_ws.col.UTCtime']-test_data.iloc[0,0]
```

```
test_data= test_data.dropna(axis=0) #결측값 삭제
test_data=test_data.astype({"_ws.col.UTCtime":'int','tcp.srcport': 'int','tcp.dstport': 'int','tcp.len': 'int','tcp.seq':'int','tcp.ack':
```

시간순으로 정렬하고 각 시간 데이터에서 가장 빠른 시간값을 뺀다.
결측값을 삭제하고 속성값 타입을 설정한다.

Test

모델에 테스트 데이터 적용

훈련 데이터와 동일하게 테스트 데이터를 그룹화한다.

```
groupset=[]
over_100_group=[]
gb = test_data.groupby(['_ws.col.Protoc
for key, group in gb:
    group = np.asarray(group)
    if len(group)>100:
        over_100_group.append(group)
    else :
        groupset.append(group)
```

```
gb = test_data.groupby(['_ws.col.Protocol','ip.src','ip.dst','tcp.dstport'])
for key, group in gb:
    group = np.asarray(group)
    if len(group)>100:
        over_100_group.append(group)
    else :
        groupset.append(group)
```

데이터를 100개로 제한하고 100개 미만인 경우 패딩 처리한다.

```
for i in range(len(over_100_group)):
    for j in range(0,len(over_100_group[i]),100):
        groupset.append(over_100_group[i][j:j+100])
```

```
for i in range(len(groupset)):
    groupset[i][:,0]=groupset[i][:,0]-groupset[i][0][0]
```

시간 데이터(millisecond)는 그룹별로 0부터 시작하게 만듦

Test

모델에 테스트 데이터 적용

Groupset에서 _ws.col.UTCtime, tcp.len ,tcp.seq, tcp.ack 속성값만 빼서 X에 넣음

```
X=[]
for i in range(len(groupset)):
    temp=np.delete(groupset[i],[1,2,3,4,5],1)
    num=100-len(temp)
    X.append(np.pad(temp,((0,num),(0,0)),'constant', constant_values=-1))
```

```
predict =model.predict(X_data)
```

```
y_pred_binary=np.around(predict)
```

테스트 데이터를 모델에 넣고 예측한다.
예측값은 연속적인 데이터 형태이므로(확률값) 0과 1로 변환한다.

```
pred_attack_ix=[]
for i in range(len(y_pred_binary)):
    if y_pred_binary[i]==1:
        pred_attack_ix.append(i)
```

Attack 데이터로 분류된 데이터(1)를 리스트에 추가한다.
(128개의 데이터를 attack 데이터로 분류함)

Test

모델에 테스트 데이터 적용

```
for i in range(len(pred_attack_ix
    print(groupset[pred_attack_ix[i]][0][1],groupset[pred_attack_ix[i]][0][2],groupset[pred_attack_ix[i]][0][3],groupset[pred_attack_i
```

```
FTP 172.16.0.1 192.168.10.50 52276 21
FTP 172.16.0.1 192.168.10.50 52278 21
FTP 172.16.0.1 192.168.10.50 52280 21
FTP 172.16.0.1 192.168.10.50 52282 21
FTP 172.16.0.1 192.168.10.50 52284 21
FTP 172.16.0.1 192.168.10.50 52286 21
FTP 172.16.0.1 192.168.10.50 52288 21
FTP 172.16.0.1 192.168.10.5
FTP 172.16.0.1 192.168.10.5
FTP 172.16.0.1 192.168.10.5
FTP 172.16.0.1 192.168.10.5
FTP 172.16.0.1 192.168.10.50 52298 21
FTP 172.16.0.1 192.168.10.50 52300 21
FTP 172.16.0.1 192.168.10.50 52302 21
FTP 172.16.0.1 192.168.10.50 52304 21
FTP 172.16.0.1 192.168.10.50 52306 21
FTP 172.16.0.1 192.168.10.50 52308 21
FTP 172.16.0.1 192.168.10.50 52310 21
```

Attack 데이터 리스트에 있는 정보 출력하기

```
import csv
```

128개->230개

```
for index in pred_attack_ix:
    for i in range(len(groupset[index])):
        wr.writerow([groupset[index][i][2],groupset[index][i][3],groupset[index][i][4],groupset[index][i][5],groupset[index][i][1]])
```

```
    i=pred_attack_ix[k]
    wr.writerow([groupset[i][0][2],groupset[i][0][3],groupset[i][0][4],groupset[i][0][5],groupset[i][0][1]])
```

Attack 데이터를 파일에 작성
(ip src, ip dst, tcp srcport, tcp dstport, protocol 정보만)

Test

result.csv

	A	B	C	D	E	F	G
1	172.16.0.1	192.168.10.50	52276	21	FTP		
2	172.16.0.1	192.168.10.50	52278	21	FTP		
3	172.16.0.1	192.168.10.50	52280	21	FTP		
4	172.16.0.1	192.168.10.50	52282	21	FTP		
5	172.16.0.1	192.168.10.50	52284	21	FTP		
6	172.16.0.1	192.168.10.50	52286	21	FTP		
7	172.16.0.1	192.168.10.50	52288	21	FTP		
8	172.16.0.1	192.168.10.50	52290	21	FTP		
9	172.16.0.1	192.168.10.50	52292	21	FTP		
10	172.16.0.1	192.168.10.50	52294	21	FTP		
11	172.16.0.1	192.168.10.50	52296	21	FTP		
12	172.16.0.1	192.168.10.50	52298	21	FTP		
13	172.16.0.1	192.168.10.50	52300	21	FTP		
14	172.16.0.1	192.168.10.50	52302	21	FTP		
15	172.16.0.1	192.168.10.50	52304	21	FTP		
16	172.16.0.1	192.168.10.50	52306	21	FTP		
17	172.16.0.1	192.168.10.50	52308	21	FTP		
18	172.16.0.1	192.168.10.50	52310	21	FTP		
19	172.16.0.1	192.168.10.50	52312	21	FTP		
20	172.16.0.1	192.168.10.50	52314	21	FTP		
21	172.16.0.1	192.168.10.50	52316	21	FTP		
22	172.16.0.1	192.168.10.50	52318	21	FTP		
23	172.16.0.1	192.168.10.50	52320	21	FTP		
24	172.16.0.1	192.168.10.50	52322	21	FTP		
25	172.16.0.1	192.168.10.50	52324	21	FTP		
26	172.16.0.1	192.168.10.50	52326	21	FTP		
27	172.16.0.1	192.168.10.50	52328	21	FTP		
28	172.16.0.1	192.168.10.50	52330	21	FTP		
29	172.16.0.1	192.168.10.50	52332	21	FTP		
30	172.16.0.1	192.168.10.50	52336	21	FTP		

Model Evaluation

model	데이터 개수	교차 검증 횟수	epoch	learning rate	정확도	F1 점수
LSTM	15	10	60	0.00001	0.9994020512945054	0.5775120893108479
CNN + LSTM	3	10	60	0.00001	0.9997021521906226	0.9568071122440266
CNN + LSTM	1	10	60	0.00001	0.9986554585049541	0.9469758812615957
outlier (lstm만)	15	10	25	0.0001	0.9994293392611271	0.619715060812464

TN	FP	FN	TP
10369	3	0	36
10367	5	0	36
10368	4	0	36
10369	3	0	36
10371	1	0	36
10370	2	0	36
10371	1	0	36
10371	1	0	36
10371	1	1	35
10371	1	8	28

=> 많은 데이터의 양으로 데이터 일부만 사용

정밀도: 0.941018767
(모델이 긍정으로 분류한 샘플 중 실제 긍정)

재현율: 0.975
(실제 긍정 샘플 중 모델이 긍정으로 분류)

Fall-out: 0.00021211
(실제 부정 샘플 중 모델이 긍정으로 잘못 분류)

- ***outlier detect***

```
from sklearn.ensemble import IsolationForest
```

```
X_total=np.reshape(X_total,(len(X_total),400))
model=IsolationForest( max_samples='auto', contamination=fl
model.fit(X_total)
score=model.decision_function(X_total)
anomaly=model.predict(X_total)
```

```
del_index=[]
for i in range(l
    if anomaly[i]
        del_index.append(i)
```

```
remove_X_total=np.delete(X_total, del_index, axis=0)
```

IsolationForest를 이용해 훈련데이터를 이상치를 제거하는 방법을 사용했으나, 데이터의 크기가 커서 제외함.

- ***가중치 초기화***

```
model = Sequential()
initializer = tf.keras.initializers.HeNormal() #가중치 초기화

model.add(Conv1D(256, 3, activation = 'relu', kernel_regularizer='l2', input_shape=(100, 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Dense(128, activation='relu', kernel_regularizer='l2', kernel_initializer=initializer))
model.add(LSTM(256))
model.add(Dense(128, activation='relu', kernel_regularizer='l2', kernel_initializer=initializer))
model.add(Dense(1, activation='sigmoid', kernel_regularizer='l2'))
```

그레디언트가 지나치게 커지거나 소실되는 것을 방지하기 위해 build_model 함수 내부에 가중치를 초기화하는 방법을 사용했으나, 교차 검증 단계별로 정확도 차이가 상이하여 제외함.