2021.02.14~3.1

KISA DATA CHALLENGE

SON SU MIN
JANG BO MIN

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

Time 속성 전처리

```
import datetime
```

각 csv 파일마다 time의 단위를 utc time을 밀리초 단위로 바꿈.

```
i=9
normal[i]['_ws.col.UTCtime']=pd.to_datetime(normal[i]['_ws.col.UTCtime'])
normal[i]['_ws.col.UTCtime']=normal[i]['_ws.col.UTCtime'].astype(np.int64)// 10**9
normal[i]['_ws.col.UTCtime']=normal[i]['_ws.col.UTCtime']-normal[i].iloc[0,0]
```

한 엑셀파일당 처음 밀리초를 0부터 시작하게 만듦.

```
normal[i]=normal[i].loc[:,"_ws.col.UTCtime": "tcp.ack"] #열자르기
normal[i] = normal[i].dropna(axis=0)
```

```
normal[i]
```

```
normal[i].to_csv("normal"+str(i)+".csv", mode='w',index = False)
```

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

Groupset 생성

In [7]:
```
#normal data를 그룹핑하고 100개가 넘는 그룹을 골라내기
groupset=[]
over_100_group=[]
for i in range(len(readdata.normal)):
    gb = readdata.normal[i].groupby(['_ws.col.Protocol','ip.src','ip.dst','tcp.srcport'])
    for key, group in gb:
        group = np.asarray(group)
        if len(group)>100:
            over_100_group.append(group)
        else :
            groupset.append(group)
```

In [8]:
```
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)
```

In [9]:
```
#normal data 100개가 넘는 그룹 100개씩 잘라서 그룹셋에 넣기
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])
```

In [10]:
```
len(groupset)
```

Out[10]: 1339219

1. *Protocol, ip.src/dst, tcp.srcport*
2. *Protocol, ip.src/dst, dstport*

데이터의 연속성을 반영한 그룹화 방식 적용.

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

배열 X 생성.

```
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))
```

각 groupset들의 헤더 제거

100-해당 그룹의 패킷 개수

-1로 패딩

X_data 로 변환.

```
X_data=np.asarray(X)
```

```
len(X_data)
```

```
1339219
```

Y_data (라벨)생성.

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

Normal이므로 '0'

```
len(Y_data)
```

```
1339219
```

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

```
groupset_a
```

```
[array([[3, 'FTP', '172.16.0.1', '192.168.10.50', 52108, 21, 14, 1, 21],
        [4, 'FTP', '172.16.0.1', '192.168.10.50', 52108, 21, 11, 15, 55],
        [4, 'FTP', '172.16.0.1', '192.168.10.50', 52108, 21, 6, 26, 78]],
       dtype=object),
 array([[87, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 1, 21],
        [87, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 17, 15, 55],
        [91, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 32, 77],
        [91, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 26, 46, 111],
        [94, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 72, 133],
        [94, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 20, 86, 167],
        [97, 'FTP', '172.16.0.1', '192.168.10.50', 52112, 21, 14, 106,
         190]], dtype=object),
 array([[87, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 1, 21],
        [87, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 20, 15, 55],
        [91, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 35, 77],
        [91, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 17, 49, 111],
        [94, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 66, 133],
        [94, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 17, 80, 167],
        [97, 'FTP', '172.16.0.1', '192.168.10.50', 52114, 21, 14, 97, 190]],
```

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

X_a 배열 생성

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

Y_a 배열 생성

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

X_attack_data, Y_attack_data 생성

```
X_attack_data=np.asarray(X_a)
Y_attack_data=np.asarray(Y_a)
```

X_data

Y_data

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

X_total 생성

```
X_total=np.concatenate((X_data,X_attack_data), axis=0)
#index : 0~891936 까지 normal, 총 891937
```

```
X_total.shape
```

```
(1339578, 100, 4)
```

Y_total 생성

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

```
Y_total.shape
```

```
(1339578,)
```

DATA PREPROCESSING

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

Train_test_split

```
: X_train, X_test, Y_train, Y_test = train_test_split(X_total,Y_total, test_size=0.2, shuffle=True, stratify=Y_total, ran
  X_val, X_test,Y_val,Y_test = train_test_split(X_test,Y_test, test_size=0.5, shuffle=True, stratify=Y_test, random_state
```

```
: print(X_train.shape,Y_train.shape,X_test.shape,Y_test.shape)
```

```
(1071643, 100, 4) (1071643,) (133956, 100, 4) (133956,)
```

Train : validation : test = 9:1:1 비율

Threshold 설정

Threshold

```
X_val_predict = model.predict(X_val,verbose=1)
```

```
predict = model.predict(X_test,verbose=1)
```

기존 모델 : (기본)0.5
-> 개선된 모델 : Roc 곡선, pr 곡선을 이용해 X_val, Y_val으로 설정

모델이 예측한 값(X_val_predict)과 실제 Y_val 비교를 통해 Roc곡선은 G-means, pr 곡선은 f-score 값이 가장 높은 임계값을 선택하도록 함.

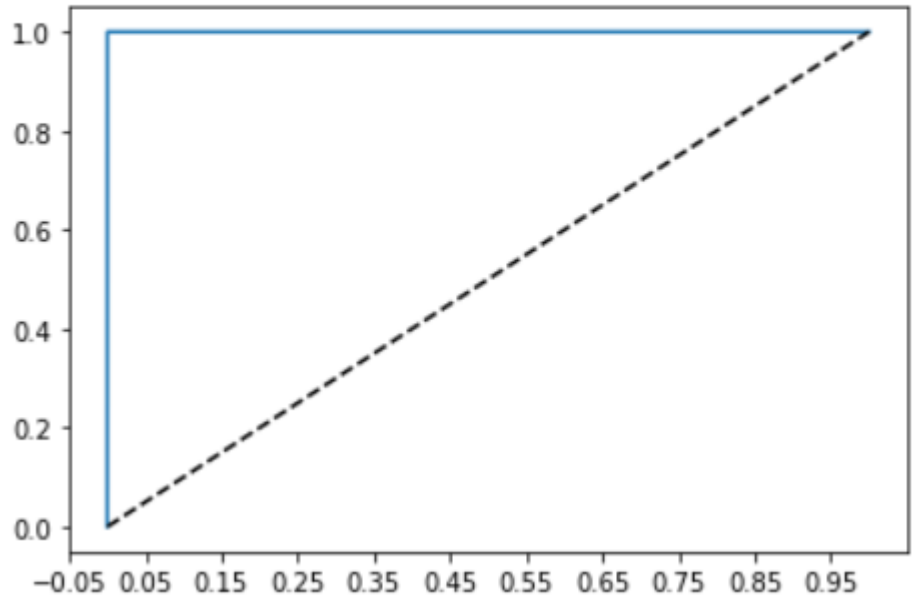
Threshold 설정

X_val_predict로 임계값 구하기

roc곡선

```
from sklearn import metrics
from sklearn.metrics import pr
from sklearn.metrics import ro

#roc 곡선
fprs, tprs, thresholds = metric

thr_index = np.arange(1, thresh

# 인덱스에 해당하는 정밀도, 재
print('thresholds : ', np.round
print('precisions : ', np.round
print('recalls : ', np.round(tp



# 그래프 그리기
def roc_curve_plot(y_real, pre
    fprs, tprs, thresholds = ro

    plt.plot(fprs, tprs, label=
    plt.plot([0,1],[0,1], 'k--'

    start, end = plt.xlim()
    plt.xticks(np.round(np.aran


roc_curve_plot(Y_val,X_val_predict)
```

```
thresholds :  [1.   1.   0.91 ... 0.   0.   0.  ]
precisions :  [0. 0. 0. ... 1. 1. 1.]
recalls :  [0.03  0.939 0.97  ... 1.    1.    1.   ]
```



Roc 그래프 그리기

rs,thresholds를 구함.

Threshold 설정


```
def optimal_threshold_roc(tpr, fpr, threshs):
    g_means = np.sqrt(tpr * (1-fpr))
    ix = np.argmax(g_means)
    print('Best Threshold=%f, G-Mean=%.3f' % (threshs[ix], g_means[ix]))
    print('Best tpr=%f, fpr=%f' % (tpr[ix], fpr[ix]))

    return threshs[ix]

threshold_roc=optimal_threshold_roc(tprs, fprs, thresholds)
threshold_roc
```

G_means값을 구하고 G_means 값들 중에서 최대값을 찾아 그 인덱스를 반환.

인덱스에 해당하는 임계값을 찾아 반환.

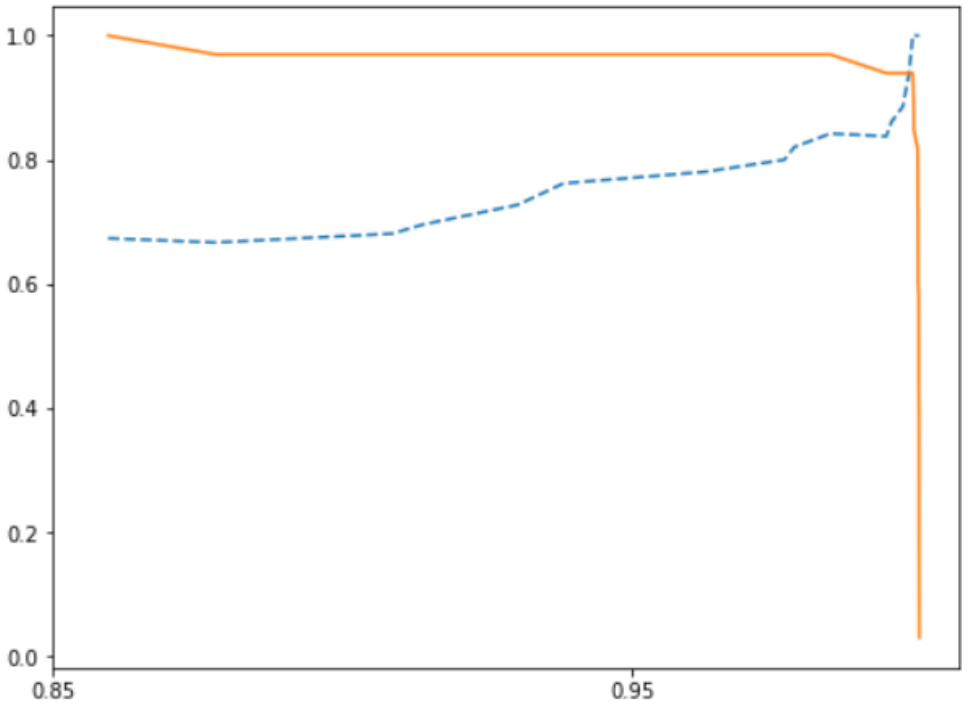
```
Best Threshold=0.859584, G-Mean=1.000
Best tpr=1.000000, fpr=0.000157

0.8595836
```

Threshold 설정

pr 곡선

```
#Y_val,X_val_predict

precisions, recalls, thresholds

# 임계값 인덱스 지정
thr_index = np.arange(0, thresho

# 인덱스에 해당하는 정밀도, 재현
print('thresholds : ', np.round(
print('precisions : ', np.round(
print('recalls : ', np.round(rec



# 그래프 그리기
def precision_recall_curve_plot
    precisions, recalls, thresho

    plt.figure(figsize=(8,6))
    threshold_boundary = thresho
    plt.plot(thresholds, precisi
    plt.plot(thresholds, recalls

    start, end = plt.xlim()
    plt.xticks(np.round(np.arang

precision_recall_curve_plot(Y_va
```

```
thresholds :  [0.86 1.   1.  ]
precisions :  [0.673 1.    1.   ]
recalls :  [1.    0.939 0.455]
```



이용해 precisions,recalls,

Threshold 설정


```
def optimal_threshold_pr(precision, recall, threshs):
    f_score = (2 * precision * recall) / (precision + recall)
    ix = np.argmax(f_score)
    print('Best Threshold=%f, F-Score=%.3f' % (threshs[ix], f_score[ix]))
    print('Best precision=%f, recall=%f' % (precision[ix], recall[ix]))

    return threshs[ix]
threshold_pr=optimal_threshold_pr(precisions, recalls, thresholds)
```

F_score값을 구하고 그 값들 중에서 최대값을 찾아 그 인덱스를 반환.

그 인덱스에 해당하는 임계값을 반환.

```
Best Threshold=0.998340, F-Score=0.969
Best precision=1.000000, recall=0.939394
```

MODEL

시도해본 것들

model 명	시간속성	그룹화 변경	교차검증	SMOTE	scaler	lstm 개수	Learning rate	epoch	tn	tp	fn	fp	맞춘개수
time_non_class_lstm1_0001_40	O	X	X	X	X	1개	0.0001	40	27	101853	7	12	29/88
									33	101829	1	36	
time_smt_lstm1_0001_60	O	X	X	O	X	1개	0.0001	60	30	101865	4	0	40개
time_smt_cross_lstm1_0001_25	O	X	O	O	X	1개	0.0001	25X4	66	203728	1	3	48개
test(roc)	O	O	X	O	X	1개	0.0001	40	35	133908	1	14	50/88 133개
test(pr)	O	O	X	O	X	1개			31	133920	5	2	30/88 73개
test_00001(roc)	O	O	X	O	X	1개	0.00001	40	35	133913	1	9	38/88 95개
test_00001(pr)	O	O	X	O	X	1개	0.00001	40	4	133922	32	0	8개
test_00001_60(pr)	O	O	X	O	X	1개	0.00001	60	28	133920	8	2	35/88 77개 4개중복→ 73개
test_00001_60(roc) 열기	O	O	X	O	X	1개	0.00001	60	36	133896	0	26	55/88 156개→144개
test_cross_25(pr)	O	O	O	O	X	1개	0.0001	25	71	267799	1	45	46/88 122개(10개중복→112개)

DATA PREPR

Need to imp

MODEL

3. Smote 적용

model 명	SMOTE	scaler	lstm 개수	Learning rate	epoch	tn	tp	fn	fp
smt_non_lstm1_0001_60	O	X	1개	0.0001	60	16	50931	1	1
non_lstm1_0001_60	X	X	1개	0.0001	60	34	101840	0	25

Class weight 보다 smote를 통해 만든 모델이 fp가 적었으며 fn은 큰 차이가 나지 않음.

따라서 smote를 적용해 모델을 만들기로 결정

MODEL

3. Smote 적용

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(random_state=0)
```

```
X_train = np.reshape(X_train,(815191,400))
```

Smote를 적용시키기 위해 2차원 배열로 변환

```
X_train_over, Y_train_over = smote.fit_sample(X_train, Y_train)

print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', X_train.shape, Y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', X_train_over.shape, Y_train_over.shape)
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(Y_train_over).value_counts())
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (815191, 400) (815191,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (1629846, 400) (1629846,)
SMOTE 적용 후 레이블 값 분포:
 1.0    814923
0.0    814923
dtype: int64
```

Smote의 적용으로 데이터의 개수가 1629846으로 증가

Smote를 적용시켜 데이터 불균형 해소

MODEL

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

3. Smote 적용

```
X_train = np.reshape(X_train,(815191,100,4))
X_train_over = np.reshape(X_train_over,(1629846,100,4))
```

```
print(X_train_over.shape, Y_train_over.shape)
```

```
(1629846, 100, 4) (1629846,)
```

다시 reshape으로 3차원으로 변경

MODEL

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

4. 교차검증

```
X_val, X_test, Y_val, Y_test = train_test_split(X_test,Y_test, test_size=0.5,shuffle=True,stratify=Y_test,random_state=101)
```

```
print(X_train_over.shape,Y_train_over.shape,X_val.shape,Y_val.shape,X_test.shape,Y_test.shape)
```

```
(1629846, 100, 4) (1629846,) (101899, 100, 4) (101899,) (101899, 100, 4) (101899,)
```

Test data를 validation과 test data로 나눔 -> smote 적용X

```
X_one,X_two,Y_one,Y_two = train_test_split(X_train_over,Y_train_over, test_size=0.5, shuffle=True, stratify=Y_train_over, random_state=101
```

```
X1, X2, Y1, Y2 = train_test_split(X_one,Y_one, test_size=0.5,shuffle=True,stratify=Y_one,random_state=101)
```

```
X3, X4, Y3, Y4 = train_test_split(X_two,Y_two, test_size=0.5,shuffle=True,stratify=Y_two,random_state=101)
```

```
print(X1.shape,X2.shape,X3.shape,X4.shape,Y1.shape,Y2.shape,Y3.shape,Y4.shape)
```

```
(407461, 100, 4) (407462, 100, 4) (407461, 100, 4) (407462, 100, 4) (407461,) (407462,) (407461,) (407462,)
```

K-fold 교차검증을 사용하기 위해 train_data_over(smote적용)를 4개로 나눔.

Stratify 옵션을 추가해 train_data set에도 정상/악성 값이 균등하게 분포되도록 함.

MODEL

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

4. 교차검증

```
        def build_model():
k = 4
all_history = []

for i in range(k):
    print('Fold #',i)
    val_data = X_data[i]
    val_target = Y_data[i]
    temp_x=np.concatenate((X_data[(i+1)//k],X_data[(i+2)//k]),axis=0)
    partial_train_data = np.concatenate((temp_x,X_data[(i+3)//k]),axis=0)

    temp_y=np.concatenate((Y_data[(i+1)//k],Y_data[(i+2)//k]),axis=0)
    partial_train_targets = np.concatenate((temp_y,Y_data[(i+3)//k]),axis=0)

                                                                                    ics=METRICS)
    model = build_model()
    history = model.fit(partial_train_data,partial_train_targets,
                        validation_data = (val_data,val_target),epochs=25, batch_size=128,verbose=1)

    all_history.append(history.history)
```

각 train이 돌아가며 validation값이 되어 교차검증을 하도록 함.

MODEL

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

4. 교차검증

model 명	시간속성	그룹화 변경	교차검증	SMOTE	scaler	lstm 개수	Learning rate	epoch	tn	tp	fn	fp	맞춘개수
time_smt_cross_lstm1_0001_25	O	X	O	O	X	1개	0.0001	25X4	66	203728	1	3	48개
test_cross_25(pr)	O	O	O	O	X	1개	0.0001	25	71	267799	1	45	46/88 122개(10개중복→112개)

교차검증은 두 모델로 평가 하였음.

-> 교차검증을 통해 만든 모델들의 test결과는 좋았지만, 실제 test data를 돌릴 때 attack data의 예측값이 현저히 작은 문제가 발생함(이전 문제점)

-> 시간 속성을 추가함으로써 해결하였음. 하지만 이번 결과에서도 다른 모델이 더 성능이 좋아 선택되지 못했음.

MODEL

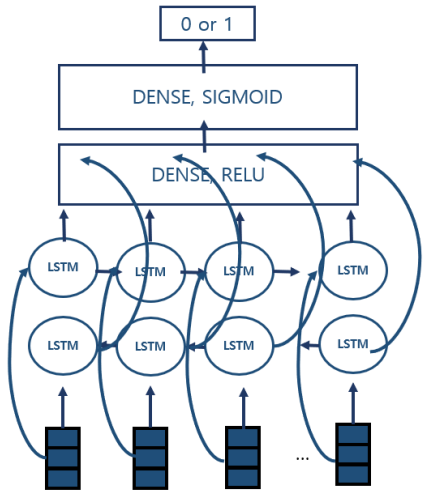
DATA PREPROCESSING
MODEL
RESULT
Need to improvement

최종 모델

Test_00001_60(roc)로 선택

model 명	시간속성	그룹화 변경	교차검증	SMOTE	scaler	lstm 개수	Learning rate	epoch	tn	tp	fn	fp	맞춘개수
test_00001_60(roc)	O	O	X	O	X	1개	0.00001	60	36	133896	0	26	55/88 156개→144개

그룹화 변경, lr=0.00001, epoch=60, roc 기준

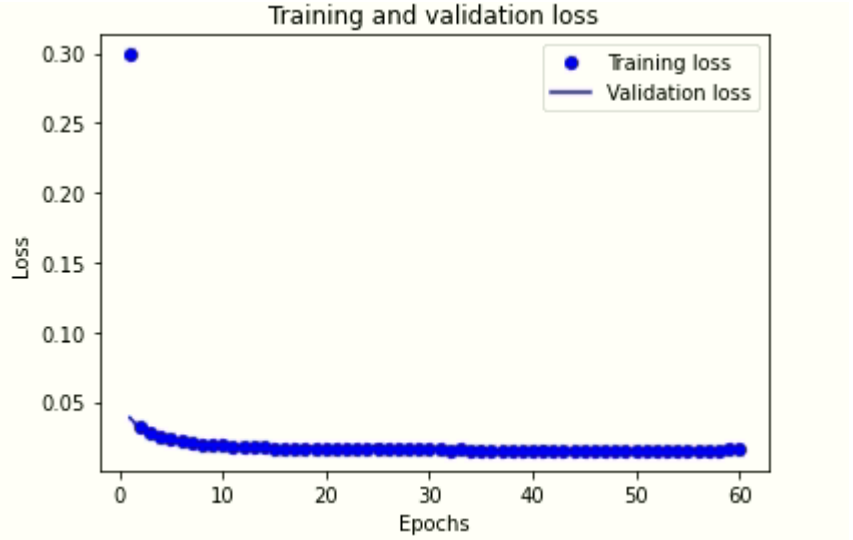



```
learning_rate = 0.00001
seq_length = 100
data_dim = 4
METRICS = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy')
]
model = Sequential()
model.add(Masking(mask_value=-1.,input_shape=(100, 4)))
model.add(Bidirectional(LSTM(128, kernel_regularizer='l2', input_shape=(100,4))))

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)

history = model.fit(X_train_over, Y_train_over,validation_data=(X_val,Y_val), epochs=60, batch_size=64)

model.save('test_00001_60.h5')
```

MODEL

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

```
learning_rate = 0.00001
seq_length = 100
data_dim = 4
METRICS = [
    tf.keras.metrics.BinaryAccuracy(name='accuracy')
]
model = Sequential()
model.add(Masking(mask_value=-1.,input_shape=(100, 4)))
model.add(Bidirectional(LSTM(128, kernel_regularizer='l2', input_shape=(100,4))))

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)

history = model.fit(X_train_over, Y_train_over,validation_data=(X_val,Y_val), epochs=60, batch_size=64)

model.save('test_00001_60.h5')
```



총 144개를 예측함. 88개의 공개된 test에서 55개를 맞춤

RESULT

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

Test 파일에 대한 예측

총 144개를 예측함. 88개의 공개된 test에서 55개를 맞춤

172.16.0.1	192.168.10	52276	21	6
172.16.0.1	192.168.10	52278	21	6
172.16.0.1	192.168.10	52280	21	6
172.16.0.1	192.168.10	52282	21	6
172.16.0.1	192.168.10	52284	21	6
172.16.0.1	192.168.10	52286	21	6
172.16.0.1	192.168.10	52288	21	6
172.16.0.1	192.168.10	52290	21	6
172.16.0.1	192.168.10	52292	21	6
172.16.0.1	192.168.10	52294	21	6
172.16.0.1	192.168.10	52296	21	6
172.16.0.1	192.168.10	52298	21	6
172.16.0.1	192.168.10	52300	21	6
172.16.0.1	192.168.10	52302	21	6
172.16.0.1	192.168.10	52304	21	6
172.16.0.1	192.168.10	52306	21	6
172.16.0.1	192.168.10	52308	21	6
172.16.0.1	192.168.10	52310	21	6
172.16.0.1	192.168.10	52312	21	6
172.16.0.1	192.168.10	52314	21	6
172.16.0.1	192.168.10	52316	21	6
172.16.0.1	192.168.10	52318	21	6
172.16.0.1	192.168.10	52320	21	6
172.16.0.1	192.168.10	52322	21	6
172.16.0.1	192.168.10	52324	21	6
172.16.0.1	192.168.10	52326	21	6
172.16.0.1	192.168.10	52328	21	6
172.16.0.1	192.168.10	52330	21	6

Need to improvement

DATA PREPROCESSING
MODEL
RESULT
Need to improvement

- scaler의 문제점을 찾고 개선하기
- 더 많은 epoch으로 테스트해보기
- 정확도 더 개선하기

2021.02.14~3.1

Thank you

SON SU MIN
JANG BO MIN