Stock Price Prediction Using Non-Transformer vs. Transformer-Based Models and Test on Developed Trading System

오유진(mydianaoh@ewhain.net)
Ewha Womans University, Dept of Artificial Intelligence

Outline

- Introduction
- Problem Definition
- Background
- Data Collection and Preprocessing
- Modeling (LSTM, D-Linear)
- Performance results (System Trading, metrics)
- Discussion & Conclusion
- Appendix

Introduction

- With the recent increase in stock price volatility worldwide, there has been a growing interest among individuals in stock market investments.
- Also, the paper **"Are Transformers Effective for Time Series Forecasting? (zeng et al., 2022)"** questions the ability of transformer-based models to capture temporal changes in stock prices. In response, the paper proposes a 'D-Linear' model as a simple baseline approach.

https://arxiv.org/pdf/2205.13504.pdf

Problem Definition

- In this project, we aim to construct both a transformer-based model (LSTM , Long Short-Term Memory) and the proposed D-Linear model. The main goal is to compare the performance of these models and evaluate the effectiveness of the approach presented in the paper.
- In summary, the primary objective of this work is to develop an effective predictive framework for stock price prediction and maximizing investment returns in developed simple trading system.

Background [Short Paper Review]

- **Are Transformers Effective for Time Series Forecasting? (zeng et al., 2022)**

1. Introduction

- 기존 TSF 문제 해결 방법론: ARIMA, GBRT, RNN, TCNs, Transformer based solutions, etc. 그중 Transformer 기반 모델들이 전통 모델들보다 LTSF 문제 해결에 앞선다고 증명하였으나 비교 대상인 non-Transformer baseline들이 autoregressive forecasting 한다는 점에서 한계가 있었음. 또한 Transformer architecture은 긴 시퀀스에서 paired elements 사이의 semantic correlations 를 추출하는 데에 용이하나 order을 고려하지 않음.
- TSF Problem Formulation :

$$\mathcal{X} = \{X_1^t, ..., X_C^t\}_{t=1}^{L} \xrightarrow{\text{predict}} \hat{\mathcal{X}} = \{\hat{X}_1^t, ..., \hat{X}_C^t\}_{t=L+1}^{L+T} \text{ at the } T \text{ future time steps.}$$

2. Transformer-Based LTSF Solutions

- Vanilla Transformer은 LTSF problem에 적용시 quadratic time/memory complexity, error 축적 문제가 발생함. 이 문제를 해결하기 위해 제시되었던 Transformer based TSF solutions :

(1) Time series decomposition (2) Input embedding strategies (3) Self-attention schemes (4) Decoders

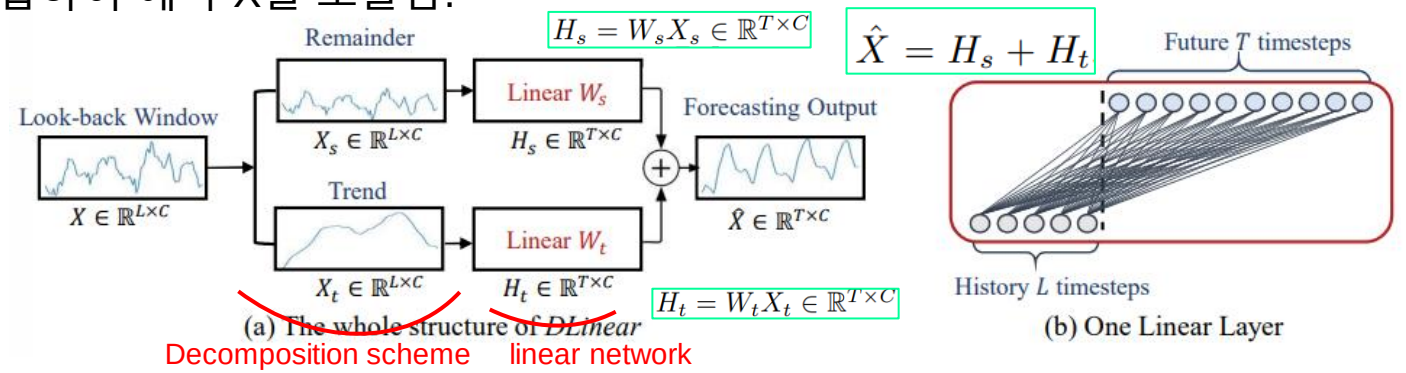
- Stock prices or electricity values 같은 raw numerical data는 point-wise semantic correlations가 거의 없음. 이 경우 order가 더 중요한데 Transformer의 self-attention은 이를 본질적으로 해결해주지 못함.

Background [Short Paper Review]

- **Are Transformers Effective for Time Series Forecasting? (zeng et al., 2022)**

3. An Embarrassingly Simple Baseline for LTSF

: 시계열 데이터에서 이동평균값을 만들고 이를 제거해 추세와 주기성(Trend, Remainder)으로 분해함. 이후 각 요소에 선형 레이어를 적용하고 합하여 예측 X를 도출함.



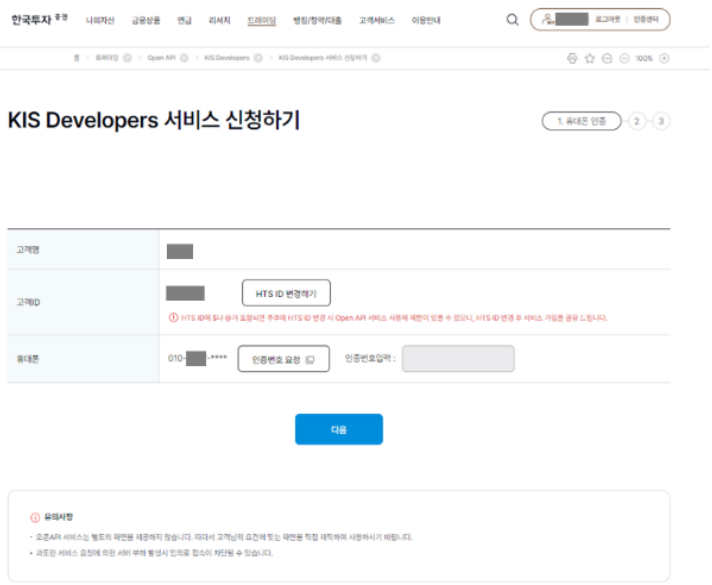
4. Experiments and Results

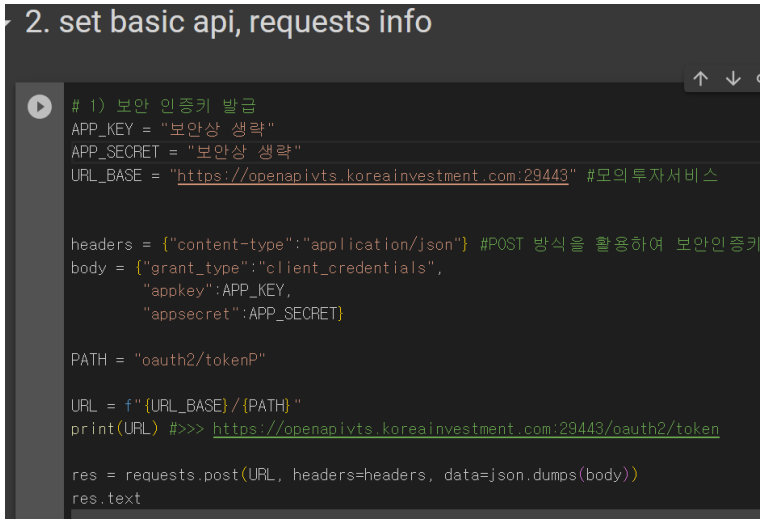
: Multiply-Accumulate Operation(MACs), 파라미터의 수, 시간 복잡도와 메모리 측면에서 본 모델이 Transformer 기반의 모델들에 비해 효율적임을 확인하였음.

Method	MACs	Parameter	Time	Memory	Time	Memory	Test Step
DLinear	0.04G	139.7K	0.4ms	687MiB	$O(L)$	$O(L)$	1
Transformer×	4.03G	13.61M	26.8ms	6091MiB	$O(L^2)$	$O(L^2)$	1
Informer	3.93G	14.39M	49.3ms	3869MiB	$O(LlogL)$	$O(LlogL)$	1
Autoformer	4.41G	14.91M	164.1ms	7607MiB	$O(LlogL)$	$O(LlogL)$	1
Pyraformer	0.80G	241.4M*	3.4ms	7017MiB	$O(L)$	$O(L)$	1
FEDformer	4.41G	20.68M	40.5ms	4143MiB	$O(L)$	$O(L)$	1

Data Collection & Preprocessing

- 한국투자증권의 트레이딩 서비스를 오픈API로 제공하는 KIS Developers를 이용하여 실시간 접근 권한 및 접근 토큰을 발급받아 SK Hynix(KRX: 000660)의 2021년 3월 2일 ~ 2023년 03월 31일까지의 데이터를 수집하였음.





Data Collection & Preprocessing

- 변수로는 stck_clpr(주식 종가), stck_oprc(주식 시가), stck_hgpr(주식 최고가), stck_lwpr(주식 최저가), stck_bsop_date(주식 영업 일자)를 사용하였음. 이때 output 변수는 stck_clpr. (feature selectio의 경우 실험적으로 위 4개를 결정하였음, 거래량 등의 변수 추가해도 큰 성능 변동x))
- 이후 'stck_bsop_date'은 컬럼명을 'date'로 변경함
- 'date'를 제외한 모든 사용하는 변수를 StandardScaler를 통해 Scaling 진행 (예측 이후 inverse_transform 으로 scaling 이전으로 되돌림. 이후 가상매매에 적용하기 위함.)
- NaN 또는 0값은 탐색 결과 존재하지 않아서 결측치 제거는 따로 이루어지지 않았음.

	stck_bsop_date	stck_clpr	stck_oprc	stck_hgpr	stck_lwpr	acml_vol	acml_tr_pbmn	flng_cls_code	prtt_rate	mod_yn	prdy_vrss_sign	prdy_vrss	rev
0	20230331	88600	89200	89500	87600	2676327	236686923700	00	0.00	N	5	-200	
1	20230330	88800	89900	90500	87700	4264354	379967206900	05	0.00	N	2	1900	
2	20230329	86900	87400	88700	86500	3070422	267552516300	00	0.00	N	5	-1500	
3	20230328	88400	85300	88700	85200	3180431	277976981700	00	0.00	N	2	2900	
4	20230327	85500	87500	87800	84700	3211190	275196944500	00	0.00	N	5	-1800	
...	...	...	...	...	...	...	...	...	...	...	...	...	
513	20210308	135500	143000	143000	135500	5587787	775259164000	00	0.00	N	5	-4500	
514	20210305	140000	138000	141500	136500	6091444	844271259500	00	0.00	N	5	-2000	
515	20210304	142000	143000	145500	139500	6586562	938397280000	00	0.00	N	5	-5000	
516	20210303	147000	142500	147000	139500	5827221	839569020500	00	0.00	N	2	2500	
517	20210302	144500	149000	150500	141000	9376523	1365279622000	00	0.00	N	2	3000	

```
df.isnull().sum()

stck_clpr    0
stck_oprc    0
stck_hgpr    0
stck_lwpr    0
date         0
dtype: int64

zero_exists = df.eq(0).any()
print(zero_exists)

stck_clpr    False
stck_oprc    False
stck_hgpr    False
stck_lwpr    False
date         False
dtype: bool
```

index	stck_clpr	stck_oprc	stck_hgpr	stck_lwpr	date
513	-1.255447624110941	-1.1398292540216084	-1.1898709795411437	-1.2402831819866322	2023-03-27 00:00:00
514	-1.0895068562391708	-1.2644980786802218	-1.1392085764046802	-1.211265337310426	2023-03-28 00:00:00
515	-1.175338287896983	-1.1454960187788181	-1.1392085764046802	-1.13581894115229	2023-03-29 00:00:00
516	-1.0666184744637541	-1.0038268998485755	-1.037883770131753	-1.0661761139293953	2023-03-30 00:00:00
517	-1.0780626653514624	-1.0434942531490434	-1.094175329172268	-1.0719796828646364	2023-03-31 00:00:00

▲Preprocessed dataframe for LSTM model

Data Collection & Preprocessing

- 전체 변수 기본 정보는 KIS Developers API 문서에서 확인 가능함.

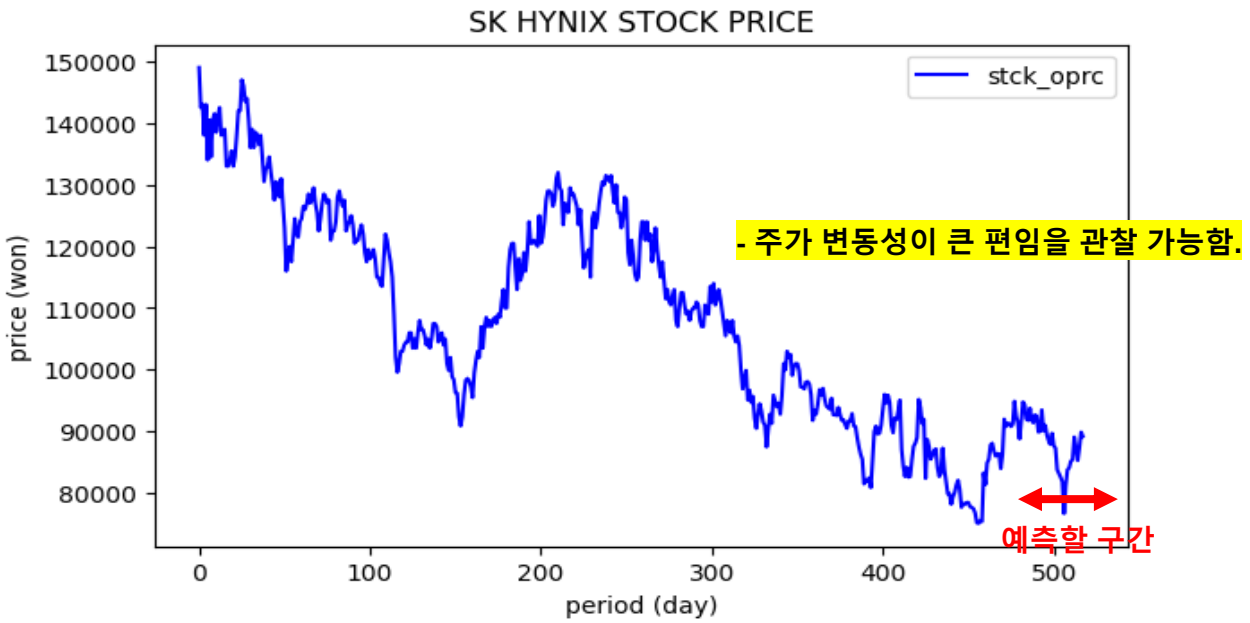
Response

</> Header

Element	한글명	Type	Required	Length	Descript
content-type	컨텐츠타입	String	Y	40	application/json; charset=ut
tr_id	거래ID	String	Y	13	요청한 tr_id
tr_cont	연속 거래 여부	String	Y	1	F or M : 다음 데이터 있음 D or E : 마지막 데이터
gt_uid	Global UID	String	Y	32	거래고유번호

</> Body

Element	한글명	Type	Required	Length	Descript
rt_cd	성공 실패 여부	String	Y	1	0 : 성공 0 이외의 값 : 실패
msg_cd	응답코드	String	Y	8	응답코드
msg1	응답메세지	String	Y	80	응답메세지
output	응답상세	Array	Y	null	
-stck_bsop_date	주식 영업 일자	String	Y	8	
-stck_oprc	주식 시가	String	Y	10	
-stck_hgpr	주식 최고가	String	Y	10	
-stck_lwpr	주식 최저가	String	Y	10	
-stck_clpr	주식 종가	String	Y	10	



Outlook of the Stock price data

https://apiportal.koreainvestment.com/apiservice/apiservice-domestic-stock-quotations#

Modeling (1): Long and Short Term Memory (LSTM)

- window_size = 40
- make_sequence_dataset()

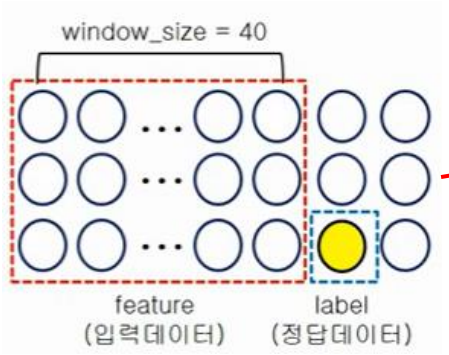
  = Created feature_list using given numpy time series data.

  * Output : X, Y , X.shape = (batch size, time steps, input dims)
- #Train data = 478, #Test data = 60, #Predicted val = 31 (= last 31 Test data)

```
[78]  feature_cols = ['stck_oprc','stck_hgpr','stck_lwpr']
      label_cols = ['stck_clpr']

      label_df = pd.DataFrame(df, columns = label_cols)
      feature_df = pd.DataFrame(df, columns = feature_cols)

      # 딥러닝 학습을 위한 numpy 변환 진행
      label_np = label_df.to_numpy()
      feature_np = feature_df.to_numpy()
```




```
[25]  def make_sequence_dataset(feature, label, window_size):
        feature_lst = []
        label_lst = []

        for i in range(len(feature) - window_size):
          feature_lst.append(feature[i: i+window_size])
          label_lst.append(label[i + window_size])

        return np.array(feature_lst), np.array(label_lst)
```

```
[26]  window_size = 40

      X, Y = make_sequence_dataset(feature_np, label_np, window_size)
      print(X.shape, Y.shape) # X.shape = (batch size, time steps, input dims)

      (478, 40, 3) (478, 1)
```

https://www.youtube.com/watch?v=LLLVYkXJw30

Modeling (1): Long and Short Term Memory (LSTM)

- **Train** : loss = ‘mae’, optimizer = ‘adam’, metrics = [‘mae’]
- EarlyStopping(monitor = ‘val_loss’, patience = 5)
- Evaluation matrix : MAE, MSE, RMSE

```python
from keras.models import Sequential
from keras.layers import Dense
import tensorflow
from keras.layers import LSTM

# LSTM 모델 구축
# 1. Sequential 모델 생성
model = Sequential()

# 2. 입력층, 은닉층 추가
model.add(LSTM(128,
               activation = 'tanh',
               input_shape = x_train[0].shape))

model.add(Dense(1, activation = 'linear'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 128)               67584

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 67,713
Trainable params: 67,713
Non-trainable params: 0
_________________________________________________________________
```

Model Architecture

Modeling (2) : D-Linear

<**Parameters**>

- seq_len = 28, label_len = 28, pred_len = 7
- individual = True (각 encoder마다 결과 구할지 유무)
- enc_in = 4 (#features = 'stck_clpr','stck_oprc','stck_hgpr','stck_lwpr')
- batch_size = 24
- test_len = 60 (#test data = 60)

<Train>

- learning_rate = {1e-4, 1e-5, 1e-6}
- Train_epochs = {50, 100}
- criterion = nn.L1Loss() , optimizer = Adam

<Evaluation matrix>

- MAE, MSE, RMSE

```
progress = tqdm(range(train_epochs))

import time
start = time.time()  # 시작 시간 저장

for epoch in progress:
    iter_count = 0
    train_loss = []
    model.train()
    epoch_time = time.time()
    for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(data_loader):
        model_optim.zero_grad()



        iter_count += 1
        batch_x = batch_x.float().to(device)

        batch_y = batch_y.float().to(device)
        batch_x_mark = batch_x_mark.float().to(device)
        batch_y_mark = batch_y_mark.float().to(device)
        # decoder input
        dec_inp = torch.zeros_like(batch_y[:, -pred_len:, :]).float()
        dec_inp = torch.cat([batch_y[:, :label_len, :], dec_inp], dim=1).float().to(device)



        outputs = model(batch_x) #, batch_x_mark, dec_inp, batch_y_mark
        f_dim = 0 #####
        outputs = outputs[:, -pred_len:, f_dim:]
        batch_y = batch_y[:, -pred_len:, f_dim:].to(device)

        loss = criterion(outputs, batch_y)
        train_loss.append(loss.item())

        loss.backward()
        model_optim.step()

    print("Epoch: {} cost time: {}".format(epoch + 1, time.time() - epoch_time))
    train_loss = np.average(train_loss)
    progress.set_description("loss: {:0.6f}".format(train_loss))
print("time :", time.time() - start)  # 현재시각 - 시작시간 = 실행 시간
torch.save(model.state_dict(), f'd_linear.pt') #trainx_y x는 훈련번호 y는 차원개수
```

Modeling (2) : D-Linear

```python
class Dataset_Pred(Dataset):
    def __init__(self, dataframe, size=None, scale=True):
        self.seq_len = size[0]
        self.label_len = size[1]
        self.pred_len = size[2]
        self.dataframe = dataframe
        self.scale = scale
        self.__read_data__()

    def __read_data__(self):
       # self.scaler = StandardScaler()  #MinMaxScaler()
        df_raw = self.dataframe
        df_raw["date"] = pd.to_datetime(df_raw["date"], format='%Y%m%d') #####

        #--------------데이터 입력 받기 전 진행하였음.-----------------
        # if self.scale:
        #   temp_df = df_raw.loc[:, ['stck_clpr', 'stck_oprc', 'stck_hgpr', 'stck_lwpr']]
        #   scaled_temp = self.scaler.fit_transform(temp_df)
        #   scaled_temp_df = pd.DataFrame(scaled_temp, columns = ['stck_clpr', 'stck_oprc', 'stck_hgpr', 'stck_lwpr'])

        # # df_raw[['stck_clpr', 'stck_oprc', 'stck_hgpr', 'stck_lwpr']] = scaled_temp_df[['stck_clpr', 'stck_oprc', 'stck_hgpr', 'stck_lwpr']]
        # df_raw = df_raw.drop(columns = ['stck_clpr', 'stck_oprc', 'stck_hgpr', 'stck_lwpr'])
        # df_raw = pd.concat([scaled_temp_df, df_raw], axis = 1)
        # print("=======Successfully Scaled=======")
        #-------------------------------------------------------------

        delta = df_raw["date"].iloc[1] - df_raw["date"].iloc[0]
        if delta>=timedelta(hours=24):
            self.freq='d'
        else:
            self.freq='h'

        border1 = 0
        border2 = len(df_raw)
        cols_data = df_raw.columns[0:]
        df_data = df_raw[cols_data]

        data = df_data.values

        datax=df_data[model_variables].values
        datay=df_data[model_output].values

        tmp_stamp = df_raw[['date']][border1:border2]
        tmp_stamp['date'] = pd.to_datetime(tmp_stamp.date)
        pred_dates = pd.date_range(tmp_stamp.date.values[-1], periods=self.pred_len+1, freq=self.freq)

        df_stamp = pd.DataFrame(columns = ['date'])
        df_stamp.date = list(tmp_stamp.date.values) + list(pred_dates[1:])
        data_stamp = time_features(df_stamp, freq=self.freq)

        self.data_x = datax[border1:border2] ##
        self.data_y = datay[border1:border2] ##

        self.data_stamp = data_stamp
```

- In order to put data into D-Linear model, additional preprocessing was needed before.
- Column ‘date’s values were object type such as 20230330, so I made Dataset_Pred class to change the values to include temporal information.

```python
        self.data_x = datax[border1:border2] ######
        self.data_y = datay[border1:border2] ########

        self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        seq_y = self.data_y[r_begin:r_end]
        seq_x_mark = self.data_stamp[s_begin:s_end]
        seq_y_mark = self.data_stamp[r_begin:r_end]
        return seq_x, seq_y, seq_x_mark, seq_y_mark

    def __len__(self):
        return len(self.data_x) - self.seq_len- self.pred_len + 1
```

```python
def time_features(dates, freq='d'):
    dates['month'] = dates.date.apply(lambda row:row.month,1)
    dates['day'] = dates.date.apply(lambda row:row.day,1)
    dates['weekday'] = dates.date.apply(lambda row:row.weekday(),1)
    dates['hour'] = dates.date.apply(lambda row:row.hour,1)
    dates['minute'] = dates.date.apply(lambda row:row.minute,1)
    dates['minute'] = dates.minute.map(lambda x:x//15)
    freq_map = {
        'y':[],'m':['month'],'w':['month'],'d':['month','day','weekday'],
        'b':['month','day','weekday'],'h':['month','day','weekday','hour'],
        't':['month','day','weekday','hour','minute'],
    }
    return dates[freq_map[freq.lower()]].values
```

Modeling (2) : D-Linear

- Notice: The baseline for the D-Linear model was obtained from a GitHub.

main ▾ LTSF-Linear / models / DLinear.py / <> Jump to ▾

mixiancmx Update scripts and models !!! ...

2 contributors

87 lines (74 sloc) | 3.58 KB

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

class moving_avg(nn.Module):
    """
    Moving average block to highlight the trend of time series
    """
    def __init__(self, kernel_size, stride):
        super(moving_avg, self).__init__()
        self.kernel_size = kernel_size
        self.avg = nn.AvgPool1d(kernel_size=kernel_size, stride=stride, padding=0)

    def forward(self, x):
        # padding on the both ends of time series
        front = x[:, 0:1, :].repeat(1, (self.kernel_size - 1) // 2, 1)
        end = x[:, -1:, :].repeat(1, (self.kernel_size - 1) // 2, 1)
        x = torch.cat([front, x, end], dim=1)
        x = self.avg(x.permute(0, 2, 1))
        x = x.permute(0, 2, 1)
        return x


class series_decomp(nn.Module):
    """
    Series decomposition block
    """
    def __init__(self, kernel_size):
```

```python
class moving_avg(nn.Module): #https://github.com/cure-lab/LTSF-Linear/blob/main/models/D
    """
    Moving average block to highlight the trend of time series
    이동평균값 만들기
    """
    def __init__(self, kernel_size, stride):
        super(moving_avg, self).__init__()
        self.kernel_size = kernel_size
        self.avg = nn.AvgPool1d(kernel_size=kernel_size, stride=stride, padding=0)

    def forward(self, x):
        # padding on the both ends of time series
        front = x[:, 0:1, :].repeat(1, (self.kernel_size - 1) // 2, 1)
        end = x[:, -1:, :].repeat(1, (self.kernel_size - 1) // 2, 1)
        x = torch.cat([front, x, end], dim=1)
        x = self.avg(x.permute(0, 2, 1))
        x = x.permute(0, 2, 1)
        return x


class series_decomp(nn.Module):
    """
    Series decomposition block
    """
    def __init__(self, kernel_size):
        super(series_decomp, self).__init__()
        self.moving_avg = moving_avg(kernel_size, stride=1)

    def forward(self, x):
        moving_mean = self.moving_avg(x)
        res = x - moving_mean
        return res, moving_mean
```

```python
# 1-layer linear network 구현 부분
class Model(nn.Module):
    """
    DLinear
    """
    def __init__(self, configs):
        super(Model, self).__init__()
        self.seq_len = configs.seq_len
        self.pred_len = configs.pred_len

        # Decomposition Kernel Size
        kernel_size = 25
        self.decomposition = series_decomp(kernel_size)
        self.individual = configs.individual
        self.channels = configs.enc_in

        if self.individual:
            self.Linear_Seasonal = nn.ModuleList()
            self.Linear_Trend = nn.ModuleList()
            self.Linear_Decoder = nn.ModuleList()
            for i in range(self.channels):
                self.Linear_Seasonal.append(nn.Linear(self.seq_len,self.pred_len))
                self.Linear_Seasonal[i].weight = nn.Parameter((1/self.seq_len)*torch.ones([self.pred_len,self.seq_len]))
                self.Linear_Trend.append(nn.Linear(self.seq_len,self.pred_len))
                self.Linear_Trend[i].weight = nn.Parameter((1/self.seq_len)*torch.ones([self.pred_len,self.seq_len]))
                self.Linear_Decoder.append(nn.Linear(self.seq_len,self.pred_len))
        else:
            self.Linear_Seasonal = nn.Linear(self.seq_len,self.pred_len)
            self.Linear_Trend = nn.Linear(self.seq_len,self.pred_len)
            self.Linear_Decoder = nn.Linear(self.seq_len,self.pred_len)
            self.Linear_Seasonal.weight = nn.Parameter((1/self.seq_len)*torch.ones([self.pred_len,self.seq_len]))
            self.Linear_Trend.weight = nn.Parameter((1/self.seq_len)*torch.ones([self.pred_len,self.seq_len]))

    def forward(self, x):
        # x: [Batch, Input length, Channel]
        seasonal_init, trend_init = self.decomposition(x)
        seasonal_init, trend_init = seasonal_init.permute(0,2,1), trend_init.permute(0,2,1)
        if self.individual:
            seasonal_output = torch.zeros([seasonal_init.size(0),seasonal_init.size(1),self.pred_len],dtype=seasonal_init.dtype).to(seasonal_init.device)
            trend_output = torch.zeros([trend_init.size(0),trend_init.size(1),self.pred_len],dtype=trend_init.dtype).to(trend_init.device)
            for i in range(self.channels):
                seasonal_output[:,i,:] = self.Linear_Seasonal[i](seasonal_init[:,i,:])
                trend_output[:,i,:] = self.Linear_Trend[i](trend_init[:,i,:])
        else:
            seasonal_output = self.Linear_Seasonal(seasonal_init)
            trend_output = self.Linear_Trend(trend_init)

        x = seasonal_output + trend_output
        return x.permute(0,2,1) # to [Batch, Output length, Channel]
```

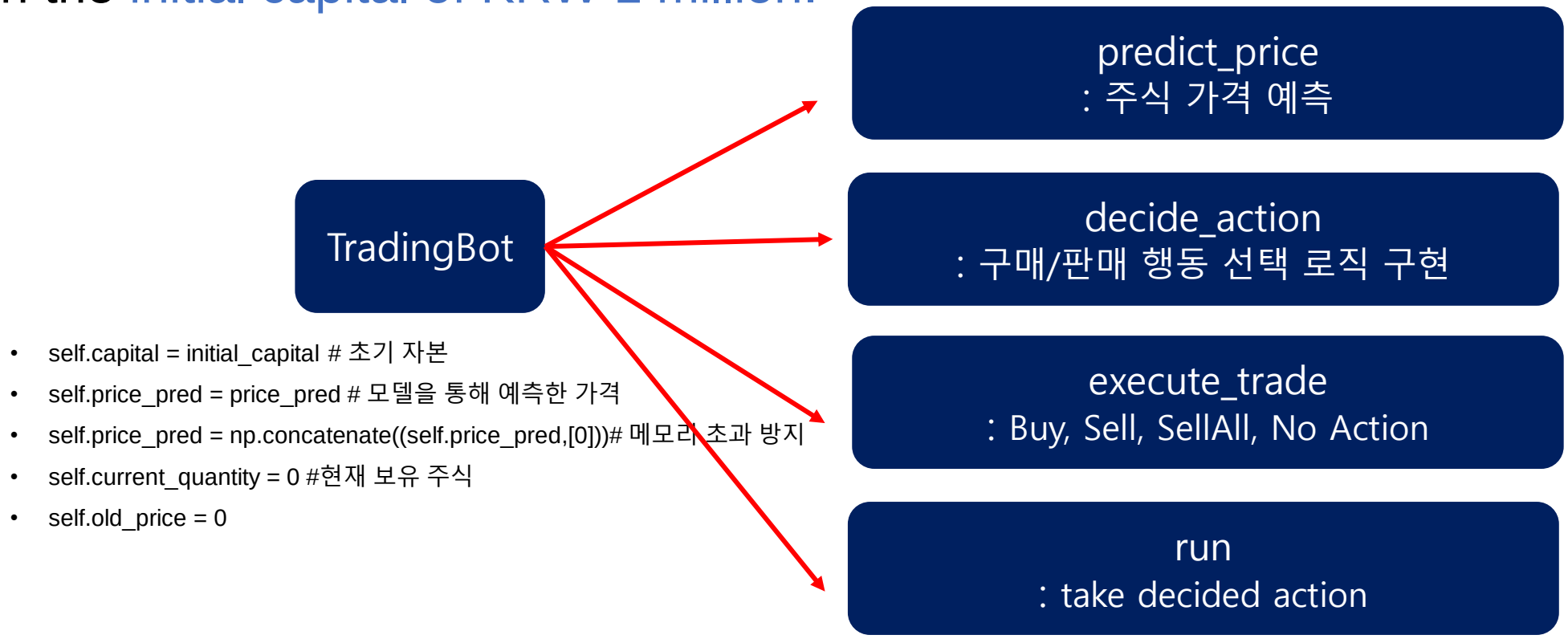
https://github.com/cure-lab/LTSF-Linear/blob/main/models/DLinear.py

Performance results (1) : metrics – System Trading

- Developed a simple trading system, TradingBot. I will conduct simulated trading using this TradingBot and observe the final capital after 30 days, starting with the initial capital of KRW 1 million.

<매매전략>

1. 종목 선정
2. 매매시점
3. 매수 거래 규칙
4. 매도 거래 규칙
5. 리스크 관리



- self.capital = initial_capital # 초기 자본
- self.price_pred = price_pred # 모델을 통해 예측한 가격
- self.price_pred = np.concatenate((self.price_pred,[0]))# 메모리 초과 방지
- self.current_quantity = 0 #현재 보유 주식
- self.old_price = 0

<Trading Bot Architecture>

Performance results (1) : metrics – System Trading

predict_price
: 주식 가격 예측

```python
def predict_price(self, current_price,day):
    # 주식 가격 예측 로직을 구현합니다.
    # 예측 결과를 반환합니다.

    print(f"day {day}'s current value: {current_price}")

    predicted_price = self.price_pred[day+1] #하루 이후의 가격 예측
    print(f"day {day+1}'s predicted value: {predicted_price}₩n")

    return predicted_price
```

decide_action
: 구매/판매 행동 선택 로직 구현

```python
def decide_action(self, previous_price, current_price, day):

    predicted_price = self.predict_price(current_price,day)

    ################ edit here ######################

    """
      구매/판매 행동을 선택할 로직을 구현합니다. ex) predicted_price가 일정 % 증가 시 구매?
    """
    if predicted_price==0: #if reach final array, sell all of the stock
        return "SellAll"

    if predicted_price > current_price or (previous_price  < current_price ) :
        return "Buy"
    elif predicted_price < current_price * 0.98 : #
        return "Sell"
    else:
        return "Hold"
```

Performance results (1) : metrics – System Trading

execute_trade
: Buy, Sell, SellAll, No Action

```python
def execute_trade(self, action, price):
    if action == "Buy":
        # 매수 로직을 구현합니다.
        # 예시로 가상의 주식을 매수하고 가상 자본을 차감합니다.


        #------------------ edit here ------------------

        quantity = 3  # 구매할 수량

        #-----------------------------------------------

        cost = price * quantity
        if self.capital >= cost:
            self.capital -= cost
            self.current_quantity+=quantity
            print(f"Bought {quantity} shares at {price}.")
        else:
            print("Not enough capital to buy.")
```

Buy

```python
elif action == "Sell":
    # 매도 로직을 구현합니다.
    # 예시로 가상의 주식을 매도하고 가상 자본을 증가시킵니다.

    if(self.current_quantity<=0): #현재 보유 개수가 0이면 판매 불가
        print("No action taken.")
    else:

        #------------------ edit here ------------------

        quantity = 1 # 판매할 수량
        self.current_quantity -= quantity

        if(self.current_quantity<0):
          print("Invalid action / revoke action")
          self.current_quantity+=1


        else:
          revenue = price * quantity
          self.capital += revenue
          print(f"Sold {quantity} shares at {price}.")
```

Sell

```python
elif action =="SellAll":
    revenue = price * self.current_quantity
    self.capital += revenue
    print(f"Sold ALL {self.current_quantity} shares at {price}.")

else:
    print("No action taken.")
```

SellAll / Else

run
: take decided action

```python
def run(self, price_real):
    for i, price in enumerate(price_real):
        if i > 0:
          action = self.decide_action(price_real[i-1], price, i)
        else:
          action = self.decide_action(price, price, i)
        self.execute_trade(action, price)
        print(f"day {i}: Current capital: {self.capital}")
        print("------------------------------₩n")
```

Performance results (2)

- **LSTM**

MAE = 0.1055198749395559

MSE = 0.016891501308593 43

RMSE = 0.1299673086148721 8

Time = 16.70s

Final Capital = 170,200 (won)

Performance results (2)

- **D-Linear #1 (lr=1e-5, epochs=50)**

  MAE = 0.17373131382941734

  MSE = 0.05406685583424251

  RMSE = 0.23252280712704831

  Time = 3.81s

  Final capital = 264,600 (won)

- **D-Linear #2 (lr=1e-4, epochs=100)**

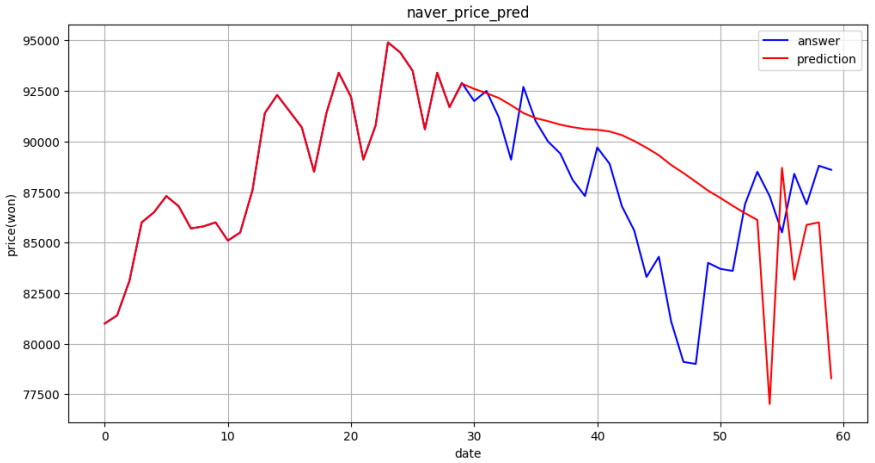
  MAE = 0.3073381714190599

  MSE = 0.1123764453209479 3

  RMSE = 0.335225961585537

  Time = 11.01s

  Final capital = 171,100 (won)



Performance results (2)

- **D-Linear #3**

(lr=1e-6, epochs=50)

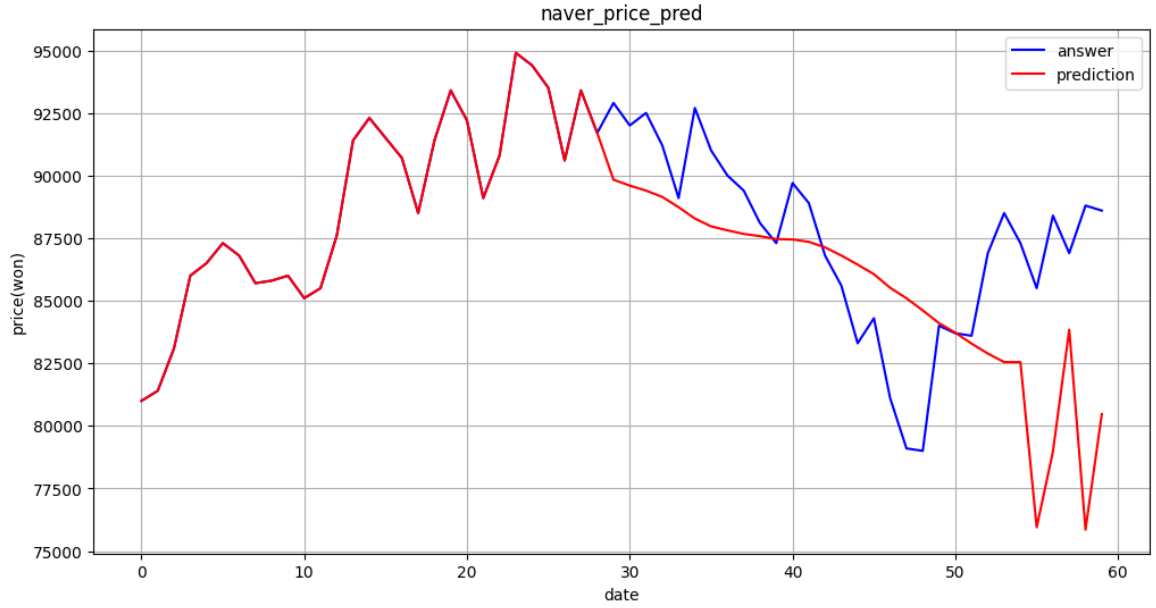
MAE = 0.1281750253934564

MSE = 0.026075247414730513

RMSE = 0.161478318714094

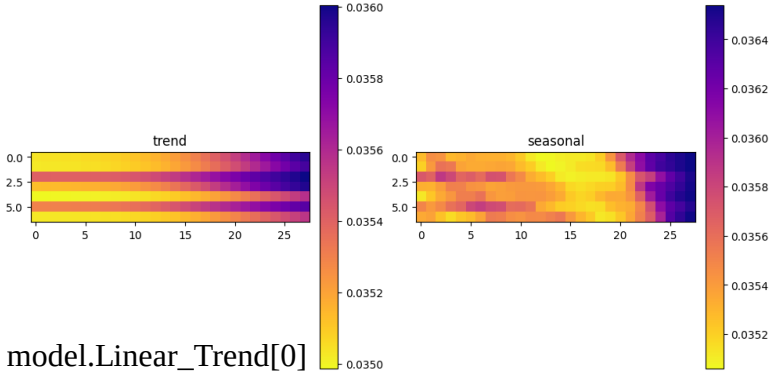
time : 4.27s

Final Capital = 81,600 (won)

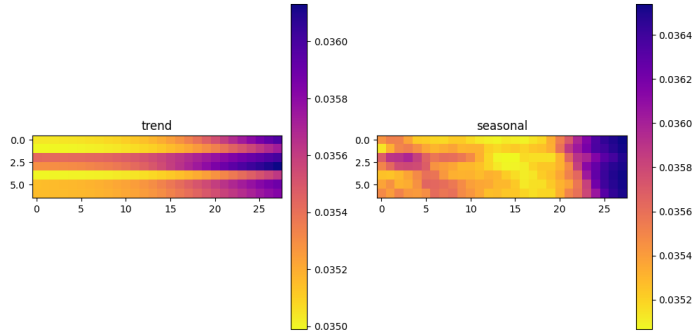


Performance results (2)

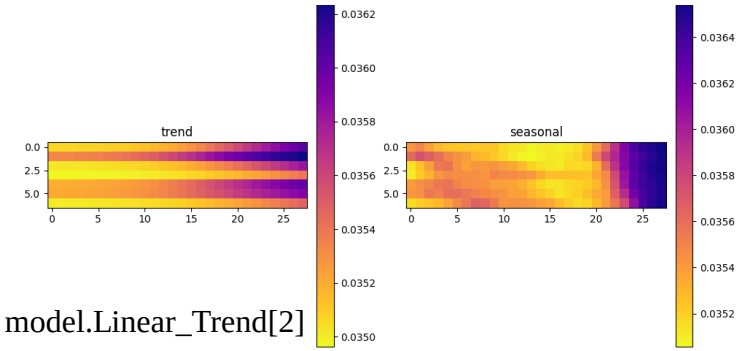
- **D-Linear #3**
  Visualization of the weights(T*L) of ***DLinear***. (Model is trained with L = 28, T = 7)
  L : a look-back window (X-axis) / T : a forecasting length(Y-axis)
- Due to the lack of periodicity and seasonality in financial data, it is hard to observe clear patterns, but the trend layer reveals greater weights of information closer to the outputs, representing their larger contributions to the predicted values. *(Are Transformers Effective for Time Series Forecasting? (zeng et al., 2022, 7pg)*



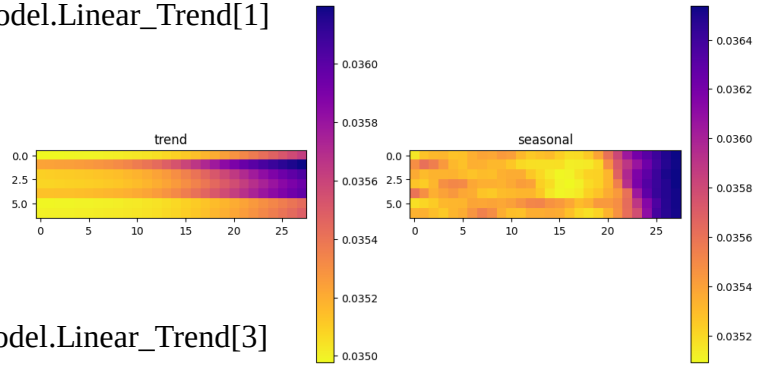
(a) model.Linear_Trend[0]



(b) model.Linear_Trend[1]



(c) model.Linear_Trend[2]



(d) model.Linear_Trend[3]

Discussion & Conclusion - Comparison

- Long-term forecasting errors in terms of MSE and MAE, the lower the better. In this experiment, LSTM showed 0.05±α lower value than D-Linear models in MAE(Mean absolute error), MSE(Mean squared error), and RMSE(Root-mean-square deviation).
- However, LSTM took longer times than all D-Linear models for training, and observing the current capital, except D-Linear #3, other two had better result than LSTM about the same TradingBot(Trading system logic). The LSTM lost almost 100,000 won more than the D-Linear #1.

Model	MAE	MSE	RMSE	Time(s)	Current Capital(won)
LSTM	0.106	0.016	0.130	16.71	170,200
D-Linear #1	0.174	0.054	0. 233	3.81	264,600
D-Linear #2	0.340	0.137	0.370	11.01	171,100
D-Linear #3	0.128	0.026	0.161	4.27	81,600

Discussion & Conclusion - Summary

- 본 연구는 전체적으로 " **Are Transformers Effective for Time Series Forecasting?** " 라는 논문을 읽고, 주가 예측에서 정말 transformer 없이 단순한 선형 모델만으로도 논문에서 제시하는 바처럼 좋은 성과를 낼 수 있을 지 확인해보는 것이었다. 구체적으로, 본 연구에서는 SK-Hynix 주식 데이터를 transformer 기반의 모델인 **LSTM**(Long and Short Term Memory)과 non-transformer 모델인 **D-Linear**을 TSF(Time Series Forcasting)에 적용하여 **주가 예측의 성능을 확인**하고, 더 나아가 구현한 단순한 Trading System 로직에서 두 모델의 예측가를 이용해 **가상 매매를 진행하고 결과값을 비교 및 평가**했다.
- D-Linear 모델에서 learning rate를 1e-4, 1e-5에서 더 줄여 **1e-6**으로 진행했을 때 가장 높은 성능(낮은 MAE, MSE, RMSE)을 보였다.(가상 매매 결과값은 가장 안 좋았음.) 비록 LSTM과 비슷한(약간 모자란) 성능이 측정되었으나, 시간적 정보(order)를 활용하고 선형 모델을 써서 시간 단축 및 가상 매매에서 전체적으로 LSTM에 비해 손실을 덜 내놓았다는 점에서 의미가 있는 모델임을 알 수 있었다. 그러나 이 경우 Trading system의 logic이 상당히 단순하게 구현되었기 때문에, 모든 모델에서 비교적 큰 손해가 생겼고 해당 logic이 D-linear 모델에 더 fit했을 수 있다는 점(특수 상황일 수 있다는 점)에서 한계가 있다고 본다.
- D-Linear model은 Long-term forecasting task에서 특히나 강력한 baseline으로 알려져 있기때문에(논문 7page), 본 연구에서 사용한 데이터셋은 비교적 short-term이었기에 LSTM보다 극적으로 좋은 결과를 내지 못한 것으로 추정해볼 수 있다. 따라서 보다 정확한 비교 및 평가를 위해서 추후 시스템 개선이 필요할 것으로 보이며, Long-term의 주가 데이터로도 연구를 진행해본다면 더욱 뚜렷한 결과를 내볼 수 있을 듯 하다.

https://arxiv.org/pdf/2205.13504.pdf

Thank you

오유진(mydianaoh@ewhain.net)
Ewha Womans University, Dept of Artificial Intelligence