1조 포트폴리오

고기원 김예은 신은아 이선기 이재연 조승기

목차

1조 포트폴리오

1. 프로젝트 소개

1-1. 프로젝트 소개 및 역할

- 이선기 (조장) : 프로젝트 총괄, NLP모델 개발, 증권 관련 도메인 조사 및 사후 원인 분석
- 고기원 : 서비스 기획, LSTM 모델 개발 및 개선, 발표 자료 제작
- 김예은 : CNN모델 개발, 증권 관련 도메인 조사 및 사후 원인 분석
- 신은아 : 서비스 기획, CNN 모델 개발, 발표 자료 제작
- 이재연 : CNN모델 개발, 서비스 웹페이지 개발
- 조승기 : LSTM모델 개발 및 CNN을 통한 개선, 서비스 웹페이지 개발

 코로나 시국 이후로 청년층의 주식 투자에 대한 관심이 급격하게 증가했다. 정보가 중요한 주식 투자에서 단기간에 투자 노하우를 쌓기에는 어려우므로, 투자에 도움이 될 수 있는 각종 지표와 정보를 제공하는 AI서비스를 개발한다.

1-2. 서비스 소개

 최종적으로 각종 지표들을 통해 학습해 익일 종가를 예측하는 모델과, 뉴스기사의 단어들을 분석해 익일 주가 등락을 간접적으로 예측할 수 있는 정보를 제공하는 모델을 배포하기로 결정되었다. 웹사이트에 모델을 구현해 주식 종목을 입력하면 익일 종가를 예측하고 관련 지표 차트 이미지를 제공한다. 더불어, 관련된 최근 기사를 자동으로 크롤링 및 분석하여 긍정과 부정을 분류한다.

2. 프로젝트 모델 개발

2-1. 초기 모델

 시계열 데이터를 학습시키기에 효과적인 기법인 LSTM 모델을 사용해서 초기 모델을 구성했다. 초기 모델은 2020년 3월 1일부터 2022년 5월 31일까지의 데이터를 갖고 학습하는 클래스를 생성했다.

- **사용한 주요 라이브러리:** finance-datareader

```
!pip -q install finance-datareader
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense, Activation, SimpleRNN
import datetime as dt
import keras
import FinanceDataReader as fdr
```

```
class MakeSet:

    def __init__(self,lst):

        if type(lst) != list:
            print('기업명을 리스트 형식으로 넣으세요.')
        else:
            self.lst = lst
            self.krx = fdr.StockListing('KRX')
            self.symbol_list = []
            self.result = []

    def check_symbol(self):
        for i in self.lst:
            self.symbol_list.append(self.krx.loc[self.krx['Name']==i,'Symbol'].values[0])
        return self.symbol_list

    def check_price(self, n=50, start_date='2020-03-01', end_date='2022-05-31'):

        seq = n
```

```
        seq_length = seq + 1


        for i in self.symbol_list:
            df = fdr.DataReader(i,start_date,end_date)['Close']

            for j in range(len(df) - seq_length):
                self.result.append(df[j:j+seq_length].to_list())

        return self.result

    def scaling(self):

        normalized_data = []
        for window in self.result:
            normalized_window = [((float(p)/float(window[0]))-1) for p in window]
            normalized_data.append(normalized_window)

        self.result = np.array(normalized_data)

        return self.result

    def train_test_set(self,n=42):

        row = int(round(self.result.shape[0]*0.9))

        x_train = self.result[:row,:-1]
        x_train = np.reshape(x_train,
                             (x_train.shape[0],x_train.shape[1],1))
        y_train = self.result[:row,-1]

        x_test = self.result[row:,:-1]
        x_test = np.reshape(x_test,
                            (x_test.shape[0],x_test.shape[1],1))
        y_test = self.result[row:,-1]

        return x_train, y_train, x_test, y_test

    def all_in_one(self,a=42,b=50,start_date='2020-03-01', end_date='2022-05-31'):
        self.check_symbol()
        self.check_price(b,start_date,end_date)
        self.scaling()

        return self.train_test_set(a)

    def data_expand(self):
        pass
```

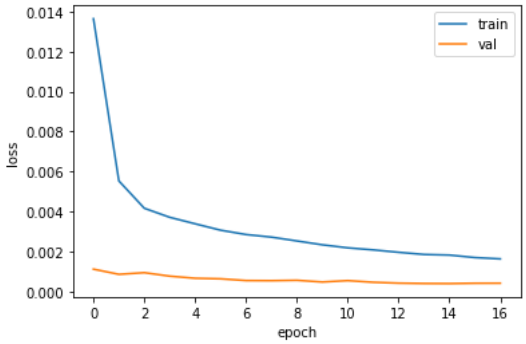
이후 성능을 확인하기 위해서 삼성전자, LG화학, 대한항공, 현대차의 데이터를 가져와 훈련 세트 총 1822개, 테스트 세트 총 202개로 구성하여 모델 성능이 괜찮은지 확인했다.

```
lst = ['삼성전자','LG화학','대한항공','현대차']
```

```
model = Sequential()

model.add(LSTM(50, return_sequences=True, input_shape=(50,1)))
# 위에서 b값을 바꿔주었으면 여기도 값을 수정해야합니다.

model.add(LSTM(64,return_sequences=True))

model.add(SimpleRNN(8))

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

model.compile(optimizer='adam', loss='mse', metrics='mae')

checkpoint_cb = keras.callbacks.ModelCheckpoint('./model/model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)

history = model.fit(x_train, y_train, epochs=100,batch_size=128,
                    validation_data=(x_test, y_test),
                    callbacks=[checkpoint_cb, early_stopping_cb]
```

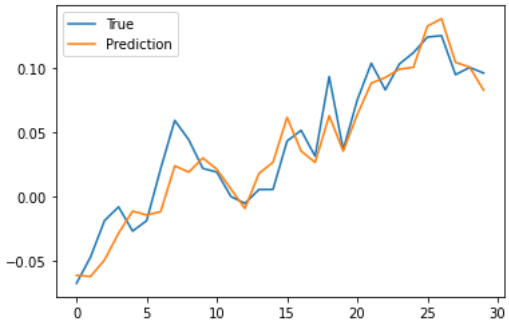
```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
plt.legend(['train', 'val'])
plt.show()
```




```
pred = model.predict(x_test)
```

```
fig = plt.figure(facecolor='white')
ax = fig.add_subplot(111)
ax.plot(y_test[-30:],label='True')
ax.plot(pred[-30:],label='Prediction')
ax.legend()
plt.show()
```



그래프를 보면 알겠지만, 해당 모델에는 한계가 있다. prediction의 결과가 하루 전날의 결과를 보여 준다. 어제의 주가와 오늘의 주가를 비슷하게 예측하는, 래깅 현상이 발생한다.

- 래깅 현상이 발생하는 이유: 주가 데이터는 시계열 데이터임에도 명확한 주기를 찾기 힘들고 외부 요인에 의한 영향이 크기 때문에 예측이 어려워진다. 이 경우 LSTM이 검증 손실을 줄이기 위해 익일 종가가 오늘의 전날과 비슷할 것으로 학습하는 것으로 보인다.

💡 LSTM 모델을 개선할 수 있는 세 가지 시도를 해 보았습니다.

2-2. 주주 차트

주식 투자 방법 중엔 오직 차트의 이미지만을 보고 매도와 매수를 결정하는 기술적 분석이라는 방법에서 시작하여 차트 이미지를 학습 시키는 CNN 모델을 생성했다.

```
import FinanceDataReader as fdr
import pandas as pd
```

학습시킨 종목은 코스피 인기 종목 200개이다. 각 종목별로 데이터프레임을 만들어서 사진을 저장시키게 유용하게 한다.

```
# 코스피 전체 종목 확인
kospi = fdr.StockListing('KOSPI')

# 네이버 주식 사이트에서 kospi 종목 크롤링한 파일 불러오기
kosp_list = pd.read_csv("./kospi.csv")

columns = ["lst_name", "lst_number"]
kosp_list = kosp_list[columns]
kosp_list = kosp_list.head(200)

# 코스피 코드
kospi_code = kosp_list["lst_number"].to_list()

# 코스피 이름
kospi_name = kosp_list["lst_name"].to_list()

# 코스피 종목별 데이터 프레임 생성
from tqdm import tqdm

for i in tqdm(kospi_code):
    globals()['df_{}'.format(i)] = fdr.DataReader('{}'.format(i), '2018-01-01')
    globals()['df_{}'.format(i)] = globals()['df_{}'.format(i)].drop(globals()['df_{}'.format(i)][globals()['df_{}'.format(i)]["Open"]
```

저장한 데이터를 캔들 스틱 차트로 생성하고 차트들을 랜덤으로 훈련 데이터와 검증용 데이터로 저장하게 한다. 이진 분류 모델을 사용할 예정이므로 훈련 데이터와 검증용 데이터 폴더에 0 (하락), 1 (상승) 폴더를 아래에 생성하여 구분한다. 차트에 나타나는 주기는 30일로 지정한다. (10일, 30일, 50일로 모델을 생성해서 결과를 봤을 때 30일이 가장 안정적이고 기폭 없는 결과를 보였다.)

```
# 캔들 차트 생성
for i, j in tqdm(zip(kospi_code, kospi_name)):


    fig, ax = plt.subplots(tight_layout=True, figsize=(1,1))
    plt.style.use(["dark_background"])
    for k in range(len(globals()['df_{}'.format(i)])-30):
        ax.axis('off')

        candlestick2_ohlc(ax, globals()['df_{}'.format(i)]['Open'].to_list()[k:k+30], globals()['df_{}'.format(i)]['High'].to_list()[k
                  globals()['df_{}'.format(i)]['Low'].to_list()[k:k+30], globals()['df_{}'.format(i)]['Close'].to_list()[k:k+30],
                  width=1, colorup='r', colordown='b')


        if globals()['df_{}'.format(i)]['Close'].to_list()[k+29] < globals()['df_{}'.format(i)]['Close'].to_list()[k+30]:
            plt.savefig('./kospi_new_30/1/{0}_{1}_{2}.png'.format(j, i, k), bbox_inches='tight',pad_inches=0, transparent=False)



        else:
            plt.savefig('./kospi_new_30/0/{0}_{1}_{2}.png'.format(j, i, k), bbox_inches='tight',pad_inches=0, transparent=False)

        plt.cla()

    plt.close(fig)
```

```
# 이미지 파일 랜덤으로 섞기

import os
dataset_0_dir = "./kospi_new_30/0/"
train_0_dir = "./kospi_0614_30/train/0/"
validation_0_dir = "./kospi_0614_30/validation/0/"
dataset_1_dir = "./kospi_new_30/1/"
train_1_dir = "./kospi_0614_30/train/1/"
validation_1_dir = "./kospi_0614_30/validation/1/"

for i in [dataset_0_dir,train_0_dir,validation_0_dir,dataset_1_dir,train_1_dir,validation_1_dir]:
    os.makedirs(i,exist_ok=True)

import random

dataset_0_lst = os.listdir("./kospi_new_30/0/")
dataset_1_lst = os.listdir("./kospi_new_30/1/")
```

```
random.seed(42)
random.shuffle(dataset_0_lst)
random.shuffle(dataset_0_lst)
random.shuffle(dataset_0_lst)
random.shuffle(dataset_0_lst)
random.shuffle(dataset_0_lst)

random.shuffle(dataset_1_lst)
random.shuffle(dataset_1_lst)
random.shuffle(dataset_1_lst)
random.shuffle(dataset_1_lst)
random.shuffle(dataset_1_lst)
```

```
# 각 폴더에 이미지 파일 랜덤으로 저장

import shutil

fnames = dataset_0_lst
for fname in fnames[:round(len(dataset_0_lst)*0.8)]:
    src = os.path.join(dataset_0_dir, fname)
    dst = os.path.join(train_0_dir, fname)
    shutil.copyfile(src, dst)

fnames = dataset_1_lst
for fname in fnames[:round(len(dataset_1_lst)*0.8)]:
    src = os.path.join(dataset_1_dir, fname)
    dst = os.path.join(train_1_dir, fname)
    shutil.copyfile(src, dst)

fnames = dataset_0_lst
for fname in fnames[round(len(dataset_0_lst)*0.8):]:
    src = os.path.join(dataset_0_dir, fname)
    dst = os.path.join(validation_0_dir, fname)
    shutil.copyfile(src, dst)

fnames = dataset_1_lst
for fname in fnames[round(len(dataset_1_lst)*0.8):]:
    src = os.path.join(dataset_1_dir, fname)
    dst = os.path.join(validation_1_dir, fname)
    shutil.copyfile(src, dst)

# 개수 확인
len(os.listdir("./kospi_0614_30/train/0/")), len(os.listdir("./kospi_0614_30/train/1/")), len(os.listdir("./kospi_0614_30/validation/0
```

훈련용 하락 데이터 81419개, 훈련용 상승 데이터 71660개, 검증용 하락 데이터 20355개, 검증용 상승 데이터 17915개를 갖고 CNN 모델을 생성해 학습시킨다.

```
import os

base_dir = './kospi_0614_30/'

# 훈련, 검증, 테스트 분할을 위한 디렉토리 설정
train_dir = os.path.join(base_dir, 'train')

train_0_dr = os.path.join(train_dir, '0')
train_1_dr = os.path.join(train_dir, '1')

validation_dir = os.path.join(base_dir, 'validation')

validation_0_dir = os.path.join(validation_dir, '0')
validation_1_dir = os.path.join(validation_dir, '1')
```

```
from keras import layers
from keras import models
from tensorflow.keras import optimizers
from keras.preprocessing.image import ImageDataGenerator

model = models.Sequential()
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu',input_shape=(72, 72, 3)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(layers.Dropout(0.3))
```

```
model.add(layers.MaxPooling2D((2, 2)))


model.add(layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

from tensorflow.keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=0.001), #default: 0.001
              metrics=['acc'])

train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(72, 72),
        color_mode = "rgb",
        batch_size=500,
        class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(72, 72),
        color_mode = "rgb",
        batch_size=500,
        class_mode='binary')

history = model.fit(train_generator,epochs=100, validation_data=validation_generator, validation_steps=70)
```

검증 정확도가 55%에서 더 이상 향상되지 않는 결과를 보입니다.

- 검증 정확도가 낮은 이유: 투자 종목을 정할 때에는 한손에는 재무제표, 한손에는 차트를 보라는 말이 있다. 그리고 거래량 또한 차트를 분석할 때 중요한 지표이다.. 이에 근거하면 다른 요소의 고려 없이 차트의 이미지만으로 주가의 등락을 예측하는 것은 당연히 검증 정확도가 낮을 수밖에 없다.

계속 50%가 넘는 예측치를 보면 주식 차트에서 패턴이란게 아예 없다고는 느껴지지 않는다. 이는 기술적 분석 또한 전혀 의미가 없는 투자 방법론이 아니라는 점을 시사한다.

2-3. 주주 주가

기존 LSTM 모델의 문제점인 전날의 주가와 비슷한 양상을 보이는 문제를 해결하고자 고려할 특성 개수를 늘려 새로운 LSTM 모델을 생성하였다. 주식 가격과 상관관계가 높은 KOSPI, ETF, 원달러 환율, 미 국채금리 네 가지 특성을 추가하여 주가가 아닌 등락을 예측하도록 클라스를 생성했다.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense, Activation, SimpleRNN
import datetime as dt

import FinanceDataReader as fdr
```

```
class MakeSet:

    def __init__(self,lst):
```

```
        if type(lst) != list:
            print('기업명을 리스트 형식으로 넣으세요.')
        else:
            self.lst = lst
            self.krx = fdr.StockListing('KRX')
            self.kospi = fdr.DataReader('KS11', '2010-03-01')['Close']
            self.wd_ratio = fdr.DataReader('USD/KRW', '2010-03-01')['Close']
            self.etf = np.array(fdr.DataReader('069500','2010-03-01')['Close'])
            self.us10yt = fdr.DataReader('US10YT=X','2010-03-01')['Close']
            self.symbol_list = []
            self.result = []
            self.arr = [[] for _ in range(4)]

    def fix_df(self):

        df_concat=pd.concat([self.kospi,self.us10yt,self.wd_ratio],axis = 1)
        df_concat.columns=["kospi","us10","usd/kr"]
        df_concat.drop(df_concat[np.isnan(df_concat['kospi'])].index,inplace=True)
        df_concat.reset_index(inplace=True)

        for i in df_concat[np.isnan(df_concat['us10'])].index:
            if i > 0:
                df_concat.loc[i,'us10'] = df_concat.loc[i-1,'us10']
            else:
                df_concat.loc[i,'us10'] = df_concat.loc[i+1,'us10']

        for i in df_concat[np.isnan(df_concat['usd/kr'])].index:
            if j > 0:
                df_concat.loc[i,'usd/kr'] = df_concat.loc[i+1,'usd/kr']
            else:
                df_concat.loc[i,'usd/kr'] = df_concat.loc[i-1,'usd/kr']

        self.wd_ratio = np.array(df_concat['usd/kr'])
        self.us10yt = np.array(df_concat['us10'])
        self.kospi = np.array(df_concat['kospi'])

        return self.kospi, self.wd_ratio, self.us10yt

    def check_symbol(self):

        for i in self.lst:
            self.symbol_list.append(self.krx.loc[self.krx['Name']==i,'Symbol'].values[0])

        return self.symbol_list

    def check_price(self, n=50, start_date='2010-03-01'):

        seq = n
        seq_length = seq + 1

        for i in self.symbol_list:
            stock = fdr.DataReader(i,start_date)['Close']
            for j in range(len(stock) - seq_length):
                self.result.append(stock[j:j+seq_length].to_list())

        for i in range(4):

            for j in range(len(self.kospi) - seq_length):
                self.arr[0].append(self.kospi[j:j+seq])

            for j in range(len(self.wd_ratio) - seq_length):
                self.arr[1].append(self.wd_ratio[j:j+seq])

            for j in range(len(self.us10yt) - seq_length):
                self.arr[2].append(self.us10yt[j:j+seq])

            for j in range(len(self.etf) - seq_length):
                self.arr[3].append(self.etf[j:j+seq])

        return self.result, self.arr

    def scaling(self):

        normalized_data = []

        for window in self.result:
            normalized_window = [((float(p)/float(window[0]))-1) for p in window]
            normalized_data.append(normalized_window)

        self.result = np.array(normalized_data)

        for i in range(len(self.arr)):

            normalized_data = []

            for window in self.arr[i]:
```

```
                normalized_window = [((float(p)/float(window[0]))-1) for p in window]
                normalized_data.append(normalized_window)

            self.arr[i] = np.array(normalized_data)


        return self.result, self.arr

    def binary_change(self):

        for i in range(len(self.result)):
            if self.result[i,-1] <= 0:
                self.result[i,-1] = 0
            else:
                self.result[i,-1] = 1

        return self.result

    def train_test_set(self,n=42):

        df = pd.DataFrame(index=range(len(self.result)),
                          columns={'f1','f2','f3','f4','f5','t'})
        df.columns = ['f1','f2','f3','f4','f5','t']


        for i in range(len(self.result)):
            df.loc[i,'f1'] = self.result[i][:-1]
            df.loc[i,'f2'] = self.arr[0][i]
            df.loc[i,'f3'] = self.arr[1][i]
            df.loc[i,'f4'] = self.arr[2][i]
            df.loc[i,'f5'] = self.arr[3][i]
            df.loc[i,'t'] = self.result[i][-1]

        df = df.sample(frac=1).reset_index(drop=True)

        row = int(round(len(self.result)*0.9))

        x_train, y_train = df.iloc[:row,:-1], df.iloc[:row,-1]
        x_test, y_test = df.iloc[row:,:-1], df.iloc[row:,-1]

        return x_train, x_test, y_train, y_test

    def all_in_one(self,a=42,b=50,start_date='2010-03-01'):

        self.check_symbol()
        self.fix_df()
        self.check_price(b,start_date)
        self.scaling()
#         self.binary_change()

        return self.train_test_set(a)

    def data_expand(self):
        pass
```

이후 성능을 확인하기 위해서 삼성전자, LG화학, 대한항공, 현대차의 데이터를 가져와 훈련 세트와 테스트 세트로 나눠서 네 가지 특성을 추가한 LSTM의 성능을 확인하였다. LSTM은 3차원 데이터를 처리하지 못하기에 머신 러닝의 보팅 기법을 사용하여 5개의 모델로 각각 학습시켰다.

```
lst = ['삼성전자','LG화학','대한항공','현대차']

x_train, x_test, y_train, y_test = MakeSet(lst).all_in_one()
```

```
x_test.reset_index(drop=True,inplace=True)
```

```
x_f1 = np.array([(x_train.loc[i,'f1']) for i in range(len(x_train))])
x_f2 = np.array([(x_train.loc[i,'f2']) for i in range(len(x_train))])
x_f3 = np.array([(x_train.loc[i,'f3']) for i in range(len(x_train))])
x_f4 = np.array([(x_train.loc[i,'f4']) for i in range(len(x_train))])
x_f5 = np.array([(x_train.loc[i,'f5']) for i in range(len(x_train))])

t_f1 = np.array([(x_test.loc[i,'f1']) for i in range(len(x_test))])
t_f2 = np.array([(x_test.loc[i,'f2']) for i in range(len(x_test))])
t_f3 = np.array([(x_test.loc[i,'f3']) for i in range(len(x_test))])
t_f4 = np.array([(x_test.loc[i,'f4']) for i in range(len(x_test))])
t_f5 = np.array([(x_test.loc[i,'f5']) for i in range(len(x_test))])
```

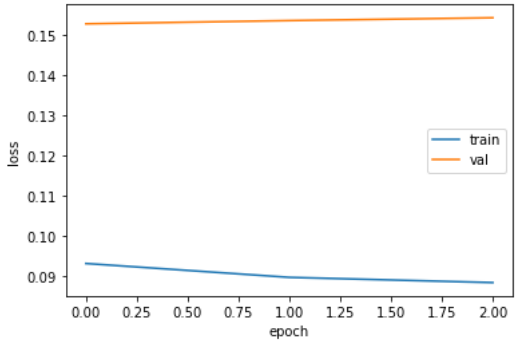
```
x_f1 = np.reshape(x_f1,
                  (x_f1.shape[0],x_f1.shape[1],1))
x_f2 = np.reshape(x_f2,
                  (x_f2.shape[0],x_f2.shape[1],1))
x_f3 = np.reshape(x_f3,
                  (x_f3.shape[0],x_f3.shape[1],1))
x_f4 = np.reshape(x_f4,
                  (x_f4.shape[0],x_f4.shape[1],1))
x_f5 = np.reshape(x_f5,
                  (x_f5.shape[0],x_f5.shape[1],1))

t_f1 = np.reshape(t_f1,
                  (t_f1.shape[0],t_f1.shape[1],1))
t_f2 = np.reshape(t_f2,
                  (t_f2.shape[0],t_f2.shape[1],1))
t_f3 = np.reshape(t_f3,
                  (t_f3.shape[0],t_f3.shape[1],1))
t_f4 = np.reshape(t_f4,
                  (t_f4.shape[0],t_f4.shape[1],1))
t_f5 = np.reshape(t_f5,
                  (t_f5.shape[0],t_f5.shape[1],1))
```

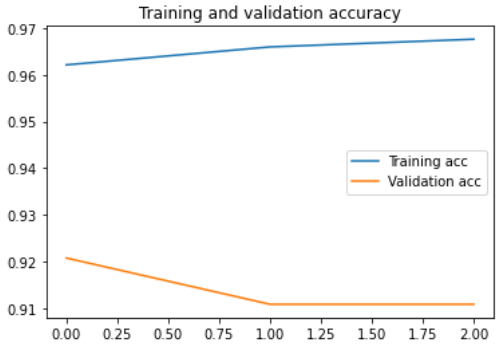
```
x_f1[0,0]
y_train,y_test = y_train.astype(int),y_test.astype(int)

# 모델 생성
model = Sequential()

model.add(LSTM(50, return_sequences=True, input_shape=(50,1)))
# 위에서 b값을 바꿔주었으면 여기도 값을 수정해야합니다.

model.add(LSTM(64,return_sequences=True))

model.add(SimpleRNN(8))

model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics='accuracy') #rmsprop
```

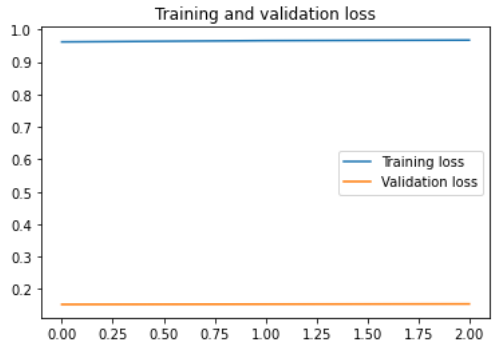
```
history1 = model.fit(x_f1,y_train,
                     validation_data=(t_f1,y_test),
#                     batch_size=10,
                     epochs=30)

history2 = model.fit(x_f2,y_train,
                     validation_data=(t_f2,y_test),
#                     batch_size=10,
                     epochs=30)

history3 = model.fit(x_f3,y_train,
                     validation_data=(t_f3,y_test),
                     epochs=30)

history4 = model.fit(x_f4,y_train,
                     validation_data=(t_f4,y_test),
#                     batch_size=10,
                     epochs=30)

history5 = model.fit(x_f5,y_train,
                     validation_data=(t_f5,y_test),
#                     batch_size=10,
                     epochs=30)
```

```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'val'])
plt.show()

plt.figure()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['accuracy']
val_loss = history.history['val_loss']
plt.plot(acc, label='Training acc')
```

```
plt.plot(val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(loss, label='Training loss')
plt.plot(val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

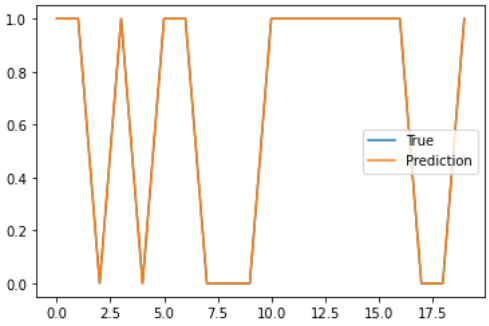







```
pred1 = model.predict(t_f1)
pred2 = model.predict(t_f2)
pred3 = model.predict(t_f3)
pred4 = model.predict(t_f4)
pred5 = model.predict(t_f5)

arr = []
for i in pred:
    if i >= 0.5:
        arr.append(1)
    else:
        arr.append(0)
arr = np.array(arr)

fig = plt.figure(facecolor='white')
ax = fig.add_subplot(111)
ax.plot(y_test[-20:],label='True')
ax.plot(arr[-20:],label='Prediction')
```

```
ax.legend()
plt.show()
```



5가지 특성을 조합하였음에도, 결국 그 결과는 주가를 통한 모델 하나만을 사용하였을 때나 차이가 없다. 보팅 기법이 아닌 하드보팅 기법을 사용해도 결과에 차이는 없다.

- 결론: 모델의 개수에 상관없이 본질 자체가 LSTM을 통해 학습한 모델이기 때문에 개선되지 않는다.

💡 3차원 이상의 데이터를 처리할 수 있는 **CNN 모델**을 사용해 본다.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense, Activation, SimpleRNN
import datetime as dt

from keras.layers import Conv2D
from keras.layers import MaxPooling2D

from keras.layers import Dense,Flatten
from keras import models

import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

from tensorflow import keras
import FinanceDataReader as fdr
```

주가의 등락을 예측할 수 있는 CNN 모델 클래스를 생성한다.

```
class MakeSet:

    def __init__(self,lst):

        if type(lst) != list:
            print('기업명을 리스트 형식으로 넣으세요.')
        else:
            self.lst = lst
            self.krx = fdr.StockListing('KRX')
            self.kospi = fdr.DataReader('KS11', '2010-03-01')['Close']
            self.wd_ratio = fdr.DataReader('USD/KRW', '2010-03-01')['Close']
            self.etf = np.array(fdr.DataReader('069500','2010-03-01')['Close'])
            self.us10yt = fdr.DataReader('US10YT=X','2010-03-01')['Close']
            self.symbol_list = []
            self.result = []
            self.arr = [[] for _ in range(4)]

    def fix_df(self):

        df_concat=pd.concat([self.kospi,self.us10yt,self.wd_ratio],axis = 1)
```

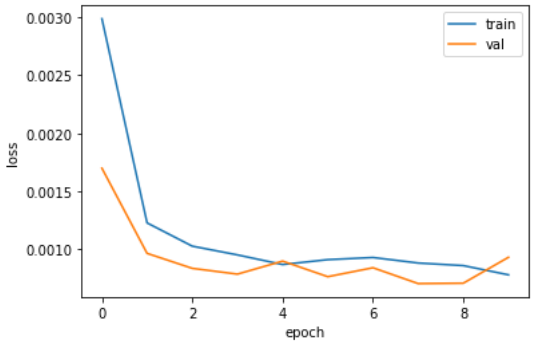
```
        df_concat.columns=["kospi","us10","usd/kr"]
        df_concat.drop(df_concat[np.isnan(df_concat['kospi'])].index,inplace=True)
        df_concat.reset_index(inplace=True)

        for i in df_concat[np.isnan(df_concat['us10'])].index:
            if i > 0:
                df_concat.loc[i,'us10'] = df_concat.loc[i-1,'us10']
            else:
                df_concat.loc[i,'us10'] = df_concat.loc[i+1,'us10']

        for i in df_concat[np.isnan(df_concat['usd/kr'])].index:
            if i > 0:
                df_concat.loc[i,'usd/kr'] = df_concat.loc[i+1,'usd/kr']
            else:
                df_concat.loc[i,'usd/kr'] = df_concat.loc[i-1,'usd/kr']

        self.wd_ratio = np.array(df_concat['usd/kr'])
        self.us10yt = np.array(df_concat['us10'])
        self.kospi = np.array(df_concat['kospi'])

        return self.kospi, self.wd_ratio, self.us10yt

    def check_symbol(self):

        for i in self.lst:
            self.symbol_list.append(self.krx.loc[self.krx['Name']==i,'Symbol'].values[0])

        return self.symbol_list

    def check_price(self, n=50, start_date='2010-03-01'):

        seq = n
        seq_length = seq + 1

        for i in self.symbol_list:
            stock = fdr.DataReader(i,start_date)['Close']
            for j in range(len(stock) - seq_length):
                self.result.append(stock[j:j+seq_length].to_list())

        for i in range(4):

            for j in range(len(self.kospi) - seq_length):
                self.arr[0].append(self.kospi[j:j+seq])

            for j in range(len(self.wd_ratio) - seq_length):
                self.arr[1].append(self.wd_ratio[j:j+seq])

            for j in range(len(self.us10yt) - seq_length):
                self.arr[2].append(self.us10yt[j:j+seq])

            for j in range(len(self.etf) - seq_length):
                self.arr[3].append(self.etf[j:j+seq])

        return self.result, self.arr

    def scaling(self):

        normalized_data = []

        for window in self.result:
            normalized_window = [((float(p)/float(window[0]))-1) for p in window]
            normalized_data.append(normalized_window)

        self.result = np.array(normalized_data)

        for i in range(len(self.arr)):

            normalized_data = []

            for window in self.arr[i]:
                normalized_window = [((float(p)/float(window[0]))-1) for p in window]
                normalized_data.append(normalized_window)

            self.arr[i] = np.array(normalized_data)


        return self.result, self.arr

#     def binary_change(self):

#         for i in range(len(self.result)):
#             if self.result[i,-1] <= 0:
#                 self.result[i,-1] = 0
#             else:
#                 self.result[i,-1] = 1

#         return self.result
```

```
    def train_test_set(self,n=42):

        df = pd.DataFrame(index=range(len(self.result)),
                          columns={'f1','f2','f3','f4','f5','t'})
        df.columns = ['f1','f2','f3','f4','f5','t']


        for i in range(len(self.result)):
            df.loc[i,'f1'] = self.result[i][:-1]
            df.loc[i,'f2'] = self.arr[0][i]
            df.loc[i,'f3'] = self.arr[1][i]
            df.loc[i,'f4'] = self.arr[2][i]
            df.loc[i,'f5'] = self.arr[3][i]
            df.loc[i,'t'] = self.result[i][-1]

#         df = df.sample(frac=1).reset_index(drop=True)

        row = int(round(len(self.result)*0.9))

        x_train, y_train = df.iloc[:row,:-1], df.iloc[:row,-1]
        x_test, y_test = df.iloc[row:,:-1], df.iloc[row:,-1]

        x_train_fix = np.array([np.array([x_train.loc[j,i] for i in x_train.columns]) for j in x_train.index]).reshape(-1,5,50,1)
        x_test_fix  = np.array([np.array([x_test.loc[j,i] for i in x_test.columns]) for j in x_test.index]).reshape(-1,5,50,1)

        y_train=y_train.astype(float)
        y_test=y_test.astype(float)

        return x_train_fix, x_test_fix, y_train, y_test

    def all_in_one(self,a=42,b=50,start_date='2010-03-01'):

        self.check_symbol()
        self.fix_df()
        self.check_price(b,start_date)
        self.scaling()

        return self.train_test_set(a)

    def data_expand(self):
        pass
```

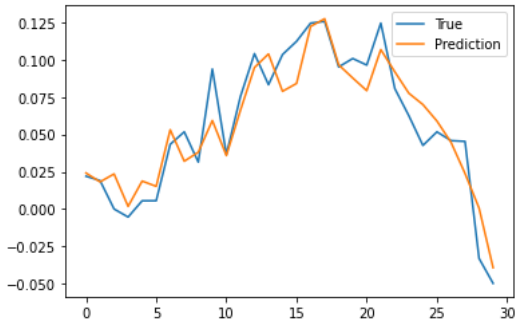
이후 성능을 확인하기 위해서 삼성전자, LG화학, 대한항공, 현대차의 데이터를 가져와 훈련 세트와 테스트 세트로 나눠서 네 가지 특성을 추가한 CNN의 성능을 확인하였다.

```
lst = ['삼성전자','LG화학','대한항공','현대차']

x_train, x_test, y_train, y_test = MakeSet(lst).all_in_one()

# 모델 생성
model = models.Sequential()
model.add(Conv2D(32, kernel_size=(5, 1), activation='relu', padding = 'same', input_shape=(5, 50, 1)))

model.add(Conv2D(64, kernel_size=(5, 1), activation='relu',padding = 'same'))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(1, activation='linear'))

model.compile(optimizer='adam', loss='mse', metrics='mae')

checkpoint_cb = keras.callbacks.ModelCheckpoint('./model/model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)

history = model.fit(x_train, y_train, epochs=20,batch_size=128,
                    validation_data=(x_test, y_test),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'val'])
plt.show()
```




```
pred = model.predict(x_test)

fig = plt.figure(facecolor='white')
ax = fig.add_subplot(111)
ax.plot(y_test.to_list()[-30:],label='True')
ax.plot(pred[-30:],label='Prediction')
ax.legend()
plt.show()
```



위의 그래프를 살펴보면, LSTM을 사용하였을 때에, 하루 전의 결과를 오늘의 결과로 하루씩 미뤄서 예측했던 경향성이 현저하게 줄었다.

- 성능을 개선할 수 있는 방법: 훈련 세트, 테스트 세트를 분리할 때 섞어 주는 작업을 시행한다.

💡 주식 가격 예측을 위해 네 가지 변수(코스피지수, 원달러환율, 미국채금리, ETF)를 사용한 이유는?

- 국내 우량주 몇 종목을 기준으로 분석해보면, 대체적으로 코스피지수, 국내 ETF와는 상관관계가 강하게 나타나는 모습을 보인다.
- 원달러 환율과는 약한 상관관계를 보이고, 미 국채 금리와는 매우 약한 상관 관계를 가진다.
- 미 국채 금리의 경우, 주식시장과는 낮은 상관관계를 보이지만 시장의 변동성이 커질 시에 그 거래량이 늘어난 다는 것에 착안하여 타 세 요소와 더불어 딥러닝 모델에 함께 학습시킨 것이다.

```
arr1 = []
arr2 = []
arr3 = []
arr4 = []

for i in range(10000):
    arr1.append(pd.DataFrame(x_train.reshape(-1,5,50)[i]).T.corr().loc[0][1])
    arr2.append(pd.DataFrame(x_train.reshape(-1,5,50)[i]).T.corr().loc[0][2])
    arr3.append(pd.DataFrame(x_train.reshape(-1,5,50)[i]).T.corr().loc[0][3])
    arr4.append(pd.DataFrame(x_train.reshape(-1,5,50)[i]).T.corr().loc[0][4])

print(f'코스피지수와의 상관계수 : {np.mean(arr1)} \n원달러환율과의 상관계수 : {np.mean(arr2)} \n미국채 금리와의 상관계수 : {np.mean(arr3)} \n국내 ETF 다
```

```
코스피지수와의 상관계수 : 0.45046491537177996
원달러환율과의 상관계수 : -0.2982738637181498
미국채 금리와의 상관계수 : 0.08833465852715022
국내 ETF 대표 종목과의 상관계수 : 0.5514653292043453
```

2-4. 주주 뉴스

```
from konlpy.tag import Okt
import pandas as pd
import numpy as np
import re

df = pd.read_csv('./dataset/pojectdata.csv')
df.head()
```

필요 모듈 및 데이터셋 로드를 한다.

	new_Date	Title	updown
0	2017-01-02	혼행족 뜨니 오픈마켓 함박웃음	1
1	2017-01-02	'최첨단 기술' 입는 패션 업체들	1
2	2017-01-02	한미약품 반면교사, 제약업계 '즉각공시' 등 풍토 변화	1
3	2017-01-02	저금리 시대...예 적금 수익률 높일 팁은?	1
4	2017-01-02	베이비붐 세대 취준생들, 세탁소 미용실 등 서비스 창업에 몰려 30%만 생존	1

'빅카인즈'에서 2017년 1월 1일부터 2022년 6월 8일까지의 경제기사 약 265만 건을 크롤링하여 다음 주식장에서 종가가 상승했다면 1을 라벨링, 반대의 경우 0을 라벨링 했다.

```
# 불용어 제거
okt = Okt()
n_ = []
title_rename = []
for i in range(len(df)):
  title_rename.append(re.sub("[\(\[].*?[\)\]]", "",df.iloc[i]['Title']))
  n_.append(' '.join(okt.nouns(df.iloc[i]['Title'])))
df['nouns'] = n_
df['Title']=title_rename
df = df[df['nouns']!='']

# 2차 불용어 제거
df['Title'] = df['Title'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
df['Title'].replace('', np.nan, inplace=True)
df = df.dropna(how='any')
df.head()
```

	new_Date	Title	updown	nouns
0	2017-01-02	혼행족 뜨니 오픈마켓 함박웃음	1	혼행 족 오픈마켓 함박 웃음
1	2017-01-02	최첨단 기술 입는 패션 업체들	1	최첨단 기술 패션 업체
2	2017-01-02	한미약품 반면교사 제약업계 즉각공시 등 풍토 변화	1	한미 약품 반면 교사 제약업 즉각 공시 등 풍토 변화
3	2017-01-02	저금리 시대예 적금 수익률 높일 팁은	1	금리 시대 예 적금 수익률 팁
4	2017-01-02	베이비붐 세대 취준생들세탁소 미용실 등 서비스 창업에 몰려 만 생존	1	베이비붐 세대 취준생들 세탁소 미용실 등 서비스 창업 만 생존

```
vocab = {}
cnt = 0
for i in df['nouns']:
    i = i.split(' ')
    for j in range(len(i)):
        if i[j] in vocab or len(i[j])<=1:
            cnt = cnt+1
```

```
            pass
        else :
            vocab[i[j]] = 0
vocab
print(len(vocab), vocab)
```

```
77823 {'혼행': 0, '오픈마켓': 0, '함박': 0, '웃음': 0, '최첨단': 0, '기술': 0, '패션': 0, '업체': 0, '한미': 0, '약품': 0, '반면': 0, '교사': 0, '제약업': 0, '즉각': 0, '공시': 0, '풍토': 0, '변화': 0, '금리': 0, '시대': 0, '적금': 0, '수익률': 0, '베이비붐': 0, '세대': 0, '취준생들': 0, '세탁소': 0, '미용실': 0, '서비스': 0, '창업': 0, '생존': 0, '단독': 0, '쌍용': 0, '건설': 0, '두바이': 0, '게이트웨이': 0, '프로젝트': 0, '수주': 0, '원점': 0, '삼육보건대': 0, '시무': 0, '개최': 0, '건양대': 0, '작업': 0, '치료': 0, '학과': 0, '국시': 0, '전국': 0, '수석': 0, '배출': 0, '한성대': 0, '아카데미': 0, '수료': 0, '매장': 0, '취업': 0, '확정': 0, '비정규직': 0, '증가': 0, '전체': 0, '근로자': 0, '동주': 0, '개교': 0, '주년': 0, '인사': 0, '전주대학교': 0, '경동대': 0, '신년': 0, '하례': 0, '안랩': 0, '연속': 0, '정규직': 0, '임금': 0, '불과': 0, '신년사': 0, '병용': 0, '사장': 0, '생산': 0, '향상': 0, '미래': 0, '성장동력': 0, '확보': 0, '김영섭': 0, '대표': 0, '연구': 0, '차별': 0, '대형': 0, '서적': 0, '도매': 0, '송인': 0, '부도': 0, '임지선': 0, '보해양조': 0, '주류': 0, '업계': 0, '바람': 0, '선도': 0, '네스프레소': 0, '디자인': 0, '캡슐': 0, '커피': 0, '머신': 0, '시티': 0, '출시': 0, '그룹': 0, '한진해운': 0, '해외': 0, '법인': 0, '인수': 0, '포기': 0, '돌발': 0, '채무': 0, '발생': 0, '원인': 0, '경남': 0, '마창대교': 0, '수익': 0, '보전': 0, '방식': 0, '변경': 0, '절감': 0, '하남': 0, '미사': 0, '강변': 0, '도시': 0, '골든': 0, '브릿지': 0, '상가': 0, '선착순': 0, '분양': 0, '유가': 0, '증권': 0, '시장': 0, '대한해운': 0, '두산건설': 0, '보령': 0, '홀딩스': 0, '상무': 0, '김정균': 0, '국민연금': 0, '리츠': 0, '개사': 0, '주식': 0, '거래': 0, '위탁': 0, '선정': 0, '대법': 0, '산자부': 0, '자료': 0, '민변': 0, '공개': 0, '신한은행': 0, '빅데이터': 0, '회의실': 0, '신설': 0, '위안화': 0, '약세': 0, '비트코인': 0, '가격': 0, '돌파': 0, '무한': 0, '경쟁': 0, '보험사': 0, '새해': 0, '상품': 0, '희망': 0, '브루나이': 0, '이순신': 0, '기재부': 0, '요즘': 0, '국제': 0, '금융': 0, '라인': 0, '연초': 0, '출장': 0, '강행': 0, '국가': 0, '인도': 0, '가늠자': 0, '외평채': 0, '발행': 0, '사활': 0, '저축은행': 0, '고무줄': 0, '대출': 0, '우리카드': 0, '할인': 0, '포인트': 0, '특화': 0, '카드': 0, '다산금융상': 0, '미래에셋': 0, '대우': 0, '성공': 0, '출범': 0, '공격': 0, '투자': 0, '글로벌': 0, '도전': 0, '현대': 0, '해상': 0, '화재보험': 0, '자산': 0, '혁신': 0, '인기': 0, '물이': 0, '국민은행': 0, '동남아': 0, '디지털': 0, '뱅크': 0, '영업': 0, '종금': 0, '진입': 0, '눈앞': 0, '최고': 0, '활용': 0, '신용': 0, '평가': 0, '모델': 0, '개발': 0, '자산운용': 0, '퇴직': 0, '연금': 0, '펀드': 0, '점유': 0, '수출': 0, '벽두': 0, '향구': 0, '수장': 0, '비씨카드': 0, '목소리': 0, '결제': 0, '한국': 0, '중소기업': 0, '서민': 0, '앞장': 0, '흑자': 0, '접목': 0, '눈길': 0, '변혁': 0, '다시': 0, '시작': 0, '노멀': 0, '신흥': 0, '자금': 0, '이탈': 0, '쓰나미': 0, '톡톡': 0, '지프': 0, '포크': 0, '베이': 0, '유로화': 0, '유럽': 0, '통합': 0, '상징': 0, '분열': 0, '씨앗': 0, '트랩': 0, '산업혁명': 0, '해법': 0, '트럼프': 0, '취임': 0, '오바마': 0, '행정부': 0, '인력': 0, '대거': 0, '고용': 0, '산업': 0, '포커스': 0, '유제품': 0, '지난해': 0, '사상': 0, '최대': 0, '원자력': 0, '회의': 0, '영기획': 0, '실장': 0, '동아': 0, '경영': 0, '정석': 0, '삼성': 0, '전자': 0, '후반': 0, '이익': 0, '예상': 0, '유명': 0, '연예인': 0, '병원': 0, '장도': 0, '사기': 0, '잠적': 0, '식당': 0, '이태원': 0, '골목': 0, '대장': 0, '장진우': 0, '거리': 0, '승부': 0, '뉴스': 0, '단신': 0, '국내': 0, '은행': 0, '가계': 0, '모두': 0, '이주열': 0, '총재': 0, '올해': 0, '경제': 0, '녹록': 0, '통화정책': 0, '상황': 0, '대응': 0, '두산': 0, '한컴': 0, '전풍': 0, '아트': 0, '라스': 0, '석준': 0, '기아': 0, '자동차': 0, '판매': 0, '두산인프라코어': 0, '임원': 0, '탄생': 0, '냉장고': 0, '미국': 0, '환경청': 0, '효율': 0, '첨단': 0, '제품': 0, '통신': 0, '워드': 0, '중국': 0, '기업': 0, '대미': 0, '직접투자': 0, '기록': 0, '이상헌': 0, '부회장': 0, '승진': 0, '테크놀로지': 0, '이랜드': 0, '신용등급': 0, '하향': 0, '조정': 0, '이종욱': 0, '대웅제약': 0, '매출': 0, '달성': 0, '재계': 0, '영화': 0, '정몽구': 0, '성장': 0, '구본무': 0, '최태원': 0, '근본': 0, '권오현': 0, '쇄신': 0, '신동빈': 0, '허창수': 0, '과감': 0, '다변': 0, '금호': 0, '사무실': 0, '액자': 0, '코오롱': 0, '배지': 0, '어가': 0, '장세욱': 0, '동국제강': 0, '직접': 0, '서울': 0, '투데이': 0, '아시아': 0, '주요': 0, '증시': 0, '소비': 0, '심리': 0, '온라인': 0, '쇼핑': 0, '올랜': 0, '박스': 0, '탈출': 0, '대동제': 0, '아포': 0, '토스': 0, '무엇': 0, '상복': 0, '부장': 0, '아토': 0, '포스': 0, '경동나비엔': 0, '보일러': 0, '세계': 0, '도약': 0, '홍수': 0, '화두':
```

```
df['updown'].value_counts()
```

```
1    1431043
0    1224857
Name: updown, dtype: int64
```

```
up = 1431043
down = 1224857
up_ratio = up/(up+down)
down_ratio = down/(up+down)

import collections
for i,w in enumerate(df['nouns']):
    w = w.split(' ')
    if (df.iloc[i]['updown']==1):
        for j in range(len(w)):
            noun = w[j]
            if len(noun)<=1:
              continue
            vocab[noun] = vocab[noun] + down_ratio
    else:
        for j in range(len(w)):
            noun = w[j]
            if len(noun)<=1:
              continue
            vocab[noun] = vocab[noun] - up_ratio

vocab
```

```
{'혼행': -0.6210655521668736,
 '오픈마켓': 32.240461613765444,
 '함박': 4.080579464588299,
 '웃음': 0.7922907488987276,
 '최첨단': 2.035256598516562,
 '기술': 113.67930607317368,
 '패션': 9.043801347939135,
 '업체': 86.85553785913527,
 '한미': 8.786104145482675,
 '약품': 65.34628073346325,
 '반면': 2.58413946308221,
 '교사': 2.34033886818034,
 '제약업': 10.646255883128198,
 '즉각': 10.985317971309 17,
 '공시': 47.68789826431243,
 '풍토': 0.45655764185398535,
 '변화': 142.246403479064,
 '금리': -256.1962163485667,
 '시대': 218.7909172031202,
 '적금': 66.02145863926081,
 '수익률': -11.593223012910528,
 '베이비붐': 0.7578688956662517,
 '세대': -83.51809405474113,
 '취준생들': 5.880824955758888,
 '세탁소': -6.3289958206257735,
 '미용실': 11.337392597612874,
 '서비스': 463.53985052175017,
 '창업': 129.02602695891895,
 '생존': 79.19391129184183,
 '단독': 28.094329982329825,
 '쌀용': 77.30847057494775,
 '건설': -105.33586731443711,
 '두바이': -7.197642983546052,
 '게이트웨이': -0.15988214917730426,
 '프로젝트': 13.2271056892172 36,
 '수주': 139.38394480218582,
 '원점': 1.0675669264656165,
 '삼육보건대': -1.7809477013441795,
```

두 글자 이상의 한국어 단어 약 78,000개를 추출했다.

단어마다 다음날 주가 등락에 따라 계수를 곱하여 더하고 빼주는 과정을 통해 감성점수를 산출했다.

```
total = []
for i,w  in enumerate(df['nouns']):
    sent_score = 0
    w= w.split(' ')
    for j in w:
        if(len(j)<=1):
          continue
        elif(j not in vocab):
          continue
        else:
          sent_score = sent_score + vocab[j]
    total.append(sent_score/len(w))
df['sent_score'] = total

df.head()
```

	new_Date	Title	updown	nouns	sent_score
0	2017-01-02	혼행족 뜨니 오픈마켓 함박웃음	1	혼행 족 오픈마켓 함박 웃음	7.298453
1	2017-01-02	최첨단 기술 입는 패션 업체들	1	최첨단 기술 패션 업체	52.903475
2	2017-01-02	한미약품 반면교사 제약업계 즉각공시 등 풍토 변화	1	한미 약품 반면 교사 제약업 즉각 공시 등 풍토 변화	29.107931
3	2017-01-02	저금리 시대에 적금 수익률 높일 팁은	1	금리 시대 에 적금 수익률 팁	2.837156
4	2017-01-02	베이비붐 세대 취준생들세탁소 미용실 등 서비스 창업에 몰려 만 생존	1	베이비붐 세대 취준생들 세탁소 미용실 등 서비스 창업 만 생존	59.988879

```
sum = 0
for i in range(len(vocab)):
  sum = sum + list(vocab.values())[i]
sent_mean = sum/len(vocab)

a_ = []
for i in range(len(df)):
  if(df.iloc[i]['sent_score']>sent_mean):
    a_.append(1)
  else:
    a_.append(0)
df['sent_label'] = a_

df
```

	new_Date	Title	updown	nouns	sent_score	sent_label
0	2017-01-02	혼행족 뜨니 오픈마켓 함박웃음	1	혼행 족 오픈마켓 함박 웃음	7.298453	1
1	2017-01-02	최점단 기술 입는 패션 업체들	1	최점단 기술 패션 업체	52.903475	1
2	2017-01-02	한미약품 반면교사 제약업계 즉각공시 등 풍토 변화	1	한미 약품 반면 교사 제약업 즉각 공시 등 풍토 변화	29.107931	1
3	2017-01-02	저금리 시대에 적금 수익률 높일 팁은	1	금리 시대 에 적금 수익률 팁	2.837156	1
4	2017-01-02	베이비붐 세대 취준생들세탁소 미용실 등 서비스 창업에 몰려 만 생존	1	베이비붐 세대 취준생들 세탁소 미용실 등 서비스 창업 만 생존	59.988879	1
...	...	...	...	...	...	...
2657196	2022-06-08	금호타이어 초등학교 교실숲 조성 기후변화 대응	0	금호 타이어 초등학교 교실 숲 조성 기후변화 대응	-7.042567	0
2657197	2022-06-08	신라면세점 공동대표에 김대중 전 현산 부문장	0	신라 면세점 대표 김대중 전 현산 부문	-54.447672	0
2657198	2022-06-08	웅진씽크빅스마트홈중학 아이패드용 나와	0	웅진 씽크빅 스마트 중학 아이패드 용	-6.543097	0
2657199	2022-06-08	경남기업 장흥역 경남아너스빌 북한산뷰 분양 흥행 이어가나	0	경남 기업 장흥역 남아 스 북한 산뷰 분양 흥행 어가	54.156230	1
2657200	2022-06-08	에스네이처 올리브영 클린뷰티 브랜드 선정	0	에스 네이처 올리브영 클린 뷰티 브랜드 선정	-34.788951	0

2655900 rows × 6 columns

기사 제목에서 감성점수의 합을 구하여 기준치(기사 감성 점수의 평균)를 넘는다면 다음날 주가에 긍정적이라는 1을 라벨링, 반대의 경우 0을 라벨링 했다.

```
from sklearn.model_selection import train_test_split

sum = 0
for i in range(len(vocab)):
  sum = sum + list(vocab.values())[i]
sent_mean = sum/len(vocab)

a_ = []
for i in range(len(df_train)):
  if(df.iloc[i]['sent_score']>sent_mean):
    a_.append(1)
  else:
    a_.append(0)
df['sent_label'] = a_

train_data, validation_data = train_test_split(df, test_size = 0.2, random_state = 42)
train_data.dropna(how='any',inplace= True)
validation_data.dropna(how='any', inplace=True)

train_data['tokenized'] = train_data['Title'].apply(okt.morphs)
validation_data['tokenized'] = validation_data['Title'].apply(okt.morphs)
X_train = train_data['tokenized'].values
Y_train = train_data['sent_label'].values
X_vali= validation_data['tokenized'].values
Y_vali = validation_data['sent_label'].values
```

```
with open('texts.pkl','wb') as f:
        pickle.dump(X_train ,f)
```

예측에 사용될 단어 리스트를 text.pkl로 저장했다.

```
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
vocab_size = 40000 # 단어집합의 크기
tokenizer = Tokenizer(vocab_size+1, oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_vali = tokenizer.texts_to_sequences(X_vali)max_len = 30
below_threshold_len(max_len, X_train)
X_train = pad_sequences(X_train, maxlen = max_len)
X_vali = pad_sequences(X_vali, maxlen = max_len)
print('전체 샘플 중 길이가 %s 이하인 샘플의 비율: %s'%(max_len, (cnt / len(nested_list))*100))
```

전체 샘플 중 길이가 30 이하인 샘플의 비율 : 99.99896456944916

토크나이징 및 패딩을 진행했다.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Embedding
from keras.layers import Bidirectional
from keras.layers import LSTM
from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(Bidirectional(LSTM(100)))
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('news_predict_v1.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, Y_train, epochs=15, callbacks=[es, mc], batch_size=256, validation_split=0.2)
```

```
Epoch 1/15
6638/6640 [============================>.] - ETA: 0s - loss: 0.1468 - acc: 0.9427
Epoch 1: val_acc improved from -inf to 0.95509, saving model to best_model.h5
6640/6640 [==============================] - 73s 11ms/step - loss: 0.1468 - acc: 0.9427 - val_loss: 0.1222 - val_acc: 0.9551
Epoch 2/15
6639/6640 [============================>.] - ETA: 0s - loss: 0.1166 - acc: 0.9572
Epoch 2: val_acc improved from 0.95509 to 0.95912, saving model to best_model.h5
6640/6640 [==============================] - 70s 11ms/step - loss: 0.1166 - acc: 0.9572 - val_loss: 0.1129 - val_acc: 0.9591
Epoch 3/15
6638/6640 [============================>.] - ETA: 0s - loss: 0.1083 - acc: 0.9610
Epoch 3: val_acc improved from 0.95912 to 0.95935, saving model to best_model.h5
6640/6640 [==============================] - 70s 11ms/step - loss: 0.1083 - acc: 0.9610 - val_loss: 0.1117 - val_acc: 0.9594
Epoch 4/15
6636/6640 [============================>.] - ETA: 0s - loss: 0.1029 - acc: 0.9634
Epoch 4: val_acc improved from 0.95935 to 0.96141, saving model to best_model.h5
6640/6640 [==============================] - 70s 10ms/step - loss: 0.1029 - acc: 0.9634 - val_loss: 0.1084 - val_acc: 0.9614
Epoch 5/15
6637/6640 [============================>.] - ETA: 0s - loss: 0.0990 - acc: 0.9653
Epoch 5: val_acc improved from 0.96141 to 0.96192, saving model to best_model.h5
6640/6640 [==============================] - 70s 10ms/step - loss: 0.0990 - acc: 0.9653 - val_loss: 0.1068 - val_acc: 0.9619
Epoch 6/15
6639/6640 [============================>.] - ETA: 0s - loss: 0.0959 - acc: 0.9666
Epoch 6: val_acc did not improve from 0.96192
6640/6640 [==============================] - 70s 11ms/step - loss: 0.0959 - acc: 0.9666 - val_loss: 0.1070 - val_acc: 0.9614
Epoch 7/15
6640/6640 [==============================] - ETA: 0s - loss: 0.0933 - acc: 0.9676
Epoch 7: val_acc did not improve from 0.96192
6640/6640 [==============================] - 70s 11ms/step - loss: 0.0933 - acc: 0.9676 - val_loss: 0.1109 - val_acc: 0.9606
Epoch 8/15
6638/6640 [============================>.] - ETA: 0s - loss: 0.0909 - acc: 0.9687
Epoch 8: val_acc did not improve from 0.96192
6640/6640 [==============================] - 69s 10ms/step - loss: 0.0909 - acc: 0.9687 - val_loss: 0.1091 - val_acc: 0.9606
Epoch 9/15
6637/6640 [============================>.] - ETA: 0s - loss: 0.0886 - acc: 0.9695
Epoch 9: val_acc did not improve from 0.96192
6640/6640 [==============================] - 69s 10ms/step - loss: 0.0886 - acc: 0.9695 - val_loss: 0.1086 - val_acc: 0.9615
Epoch 9: early stopping
```

```
model = keras.models.load_model('news_predict_v1.h5')
model.evaluate(X_vali,Y_vali)
```

```
16600/16600 [==============================] - 65s 4ms/step - loss: 0.1063 - acc: 0.9618
[0.10627663135528564, 0.9618151783943176]
```

Bi - LSTM을 통해 모델 학습, 검증 결과 val_acc : 0.96

모델 채택하고 class로 만들어 django 내에 저장했다.

```
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow import keras
import re
from konlpy.tag import Okt
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
import requests
import pandas as pd
```

```
# 사이트 시작하자마자 메모리에 업로드해야 하는 코드

import pickle
with open('texts.pkl','rb') as a:
    texts = pickle.load(a)


vocab_size = 40000
tokenizer = Tokenizer(vocab_size+1, oov_token = 'OOV')
tokenizer.fit_on_texts(texts)
texts = tokenizer.texts_to_sequences(texts)
```

```
class NewsAnalysis:

    def __init__(self,search_word):

        self.client_id = "네이버 OPEN API ID" #1.에서 취득한 아이디 넣기
        self.client_secret = "네이버 OPEN API secret"  #1.에서 취득한 키 넣기
        self.search_word = search_word #검색어
        self.encode_type = 'json' #출력 방식 json 또는 xml
        self.max_display = 100 #출력 뉴스 수
        self.sort = 'date' #결과값의 정렬기준 시간순 date, 관련도 순 sim
        self.start = 1 # 출력 위치
        self.url = f"https://openapi.naver.com/v1/search/news.{self.encode_type}?query={search_word}&display={str(int(self.max_display


        #헤더에 아이디와 키 정보 넣기 & HTTP요청 보내기

    def requestHttp(self):

        self.headers = {'X-Naver-Client-Id' : self.client_id,
                   'X-Naver-Client-Secret': self.client_secret
                   }

        self.r = requests.get(self.url, headers=self.headers)
        #요청 결과 보기 200 이면 정상적으로 요청 완료
        return self.r

    def clean_html(self,x):
        self.x = re.sub("\&\w*\;","",x)
        self.x = re.sub("<.*?>","",self.x)
        return self.x


    def to_dataframe_re_apply(self):
        self.df = pd.DataFrame(self.requestHttp().json()['items']) # 데이터 프레임으로 만들기
        self.df['title'] = self.df['title'].apply(lambda x: self.clean_html(x))
        self.df['description'] = self.df['description'].apply(lambda x: self.clean_html(x))
        return self.df



    def news_predict(self):
        okt = Okt()
        model = keras.models.load_model('news_predict_v1.h5')
        stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '걍', '과', '도', '를', '으로', '자', '에', '와', '한', '하다']
        morphed = []
        dataframe = self.to_dataframe_re_apply()
        for i in range(len(dataframe)):
            new_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣 ]','', dataframe['title'][i])
            new_sentence = re.sub("[\(\[].*?[\)\]]",'', new_sentence)
            new_sentence = okt.morphs(new_sentence, stem=True) # 토큰화
            new_sentence = [word for word in new_sentence if not word in stopwords] # 불용어 제거
            encoded = tokenizer.texts_to_sequences([new_sentence]) # 정수 인코딩
            pad_new = pad_sequences(encoded, maxlen = 30)# 패딩
            score = float(model.predict(pad_new,verbose=0)) # 예측
            if (score > 0.5):
#                 print(dataframe['title'][i])
#                 print("{0:}번 째 기사는 {1:.2f}% 확률로 긍정적 기사입니다.".format((i+1), (score * 100)))
                morphed.append('긍정적인 기사')
            else :
#                 print(dataframe['title'][i])
#                 print("{0:}번 째 기사는 {1:.2f}% 확률로 부정적 기사입니다.".format((i+1), (1 - score) * 100))
                morphed.append('부정적인 기사')
        dataframe['Prediction'] = morphed


        return dataframe['Prediction'].value_counts()
```

3. 배포

- 상단에 주식 종목을 입력하면 지난 종가와 예상 종가가 보일 수 있게 기획했다.
- 사이트 왼쪽 상단에는 날짜별 주가, KOSPI, ETF, 원/달러 환율, 미 국채 10년 금리 수치를 확인할 수 있는 표가 있다.
- 사이트 왼쪽 아래 부분에는 입력한 주식 종목에 대한 주가와 KOSPI 지수, ETF, 환율 그래프를 확인할 수 있다.
- 사이트 오른쪽 부분에서는 상단에 입력한 주식 종목에 대한 관련 기사 목록을 확인할 수 있는데, 긍정적인 기사 개수와 부정적인 기사 개수를 명시해 줘서 해당 주식에 대한 긍정 부정 정도를 한눈에 쉽게 확인할 수 있다.

> 💡 **시크릿주주가 다른 주식 정보 사이트보다 유용한 이유**

- 시크릿 주주는 여러가지 지표를 해석할 때 과거 데이터만을 이용하므로 개인의 주관에 의한 비합리적인 선택을 방지할 수 있다. 이성적인 투자자의 투자 노하우를 학습한 서비스라고 볼 수 있다.
- 최근 경제지표를 바탕으로 입력한 주식의 익일 종가를 예측한다.
- 해당 주식과 관련 기사들을 검색해 준다.
- 기사 제목의 단어들을 분석해 기사가 익일 주가에 긍정적일지 부정적일지 분류한다.

4. 결론

- 결과물에 있어서 프로젝트 기간 중 모델 테스트를 진행한 6월 셋째 주는 전세계적인 경제 악화로 주식 시장이 계속해서 하락세를 보여 하락과 상승의 결과를 골고루 검증할 수 없던 점이 아쉬웠다.
- 그리고 정말 많은 요소가 고려되어야 하는 주식시장인데, 짧은 프로젝트 기간 동안 주가를 예측하는 AI를 만드려 했다는 것이 오만했다는 생각도 들었다.
- 다만, 소규모 프로젝트의 얕은 지식과 역량으로도 노력과 연구를 통해 AI의 성능을 올릴 수 있었던 것을 생각하면, 언젠가는 거듭된 연구를 통해 AI가 주식시장의 광기마저도 정복하는 날이 올 수 있을 것이라 생각한다.

5. 개발 후기

고기원

주린이의 입장에서 처음에는 접근하기 조금 어려운 주제였지만, 정말 능력자인 팀원들 덕분에 이 프로젝트를 함께 할 수 있었던 것 같다. 직접 배운 내용을 토대로 모델을 만드는 과정에서 쉽지 않은 순간들도 있었지만 , 환상의 팀워크를 보여준 팀원들에게 정말 감사하다.

김예은

이번 프로젝트를 한 단어로 표현하면 '고난'인 것 같다. 경제 전공자로서 주식에 관련된 자세한 내용은 모르지만 적어도 사람들이 투자할 때 차트를 보고 분석한다는 정도의 지식과 CNN 모델이 관련 논문이 많아서 성능이 괜찮은 모델이 나올 줄 알았는데 생각보다 안 좋은 성능의 모델이 나와서 실망했다. 하지만 팀원분들 중에 대단한 아이디어로 성능을 높인 모델의 결과를 보면서 발상의 전환을 배울 수 있었다. 또한 같이 CNN 모델 만든 분들한테도 모델의 성능을 높일 수 있는 다양한 기법과 끈기를 본받을 수 있는 기회라서 의미 있었던 것 같다.

한 가지 아쉬웠던 점은 전공자이지만 도메인 지식이 부족해서 좀 더 쉽고 재미있게 주식 이야기를 풀어나가는 데 도움이 되지 못한 것 같은 부분이다. 하지만 이번 프로젝트를 통해서 단기간에 해당 도메인 지식을 복습하는 시간을 통해 부족한 부분을 깨달았고 성장할 수 있을 것으로 기대한다.

신은아

'머신러닝', '주가예측', '모델링' … 인생에서 이렇게 많이 듣고 고민해본 적 없는 것 같다. 수업을 들으면서도 배운걸 제대로 이해하고 있는지, 프로젝트를 진행하면서 따라갈 수 있을지 걱정이 앞섰지만, 팀원들을 잘 만난 덕분에 낙오되지 않고 쫓아갈 수 있었다. 팀원들이 아니였다면 진작에 포기할 수 있는 힘든 시간이었지만 끝까지 격려해주어 고맙고, 견디고 있는 나.. 대견하다.

이선기

주제도 어렵고 작업량도 많았지만 같이 함께해준 조원들이 너무 고맙다.

그리고 LSTM배우고 주가를 예측하려는 사람이 다시는 없길 바란다....

이재연

너무 수고한 조원들에게 감사하는 마음입니다

배웠던 내용으로 주식 차트를 예측한다는 것이 매우 흥미로웠다.

CNN으로 꽤 높은 정확도로 예측할 줄 알았는데 생각보다 고려해야 하는 점이 많았고, 진행할수록 도메인 지식의 중요성을 다시 한번 깨닫게 되었다. 이런저런 시도를 많이 하면서 코드를 다루는 게 편해졌고, 빅 데이터를 다루는 과정에서 제한된 환경 (메모리, 그래픽카드 유무)을 극복하는 것 또한 성장으로 이끌었던 것 같다. 팀원들 넘 고생쓰~~

조승기

CNN을 통해 모델 개선을 시도하였고, 어느 정도는 만족스러운 부분도 있었다. 하지만 여전히 아쉬운 부분이 남아있다. 더 많은 피쳐를 넣음으로써 모델을 개선할 수 있는 여지가 남아 있는 것과, 또한 종목에 따라 편차가 심하게 나는 부분 등이 그 예이다.

아쉬운 점은 여전히 남아있지만, 그럼에도 다양한 시도를 해볼 수 있었으며 앞으로 무엇을 공부해야 할 지 깨달았다는 점에서 의미 있는 프로젝트였다.