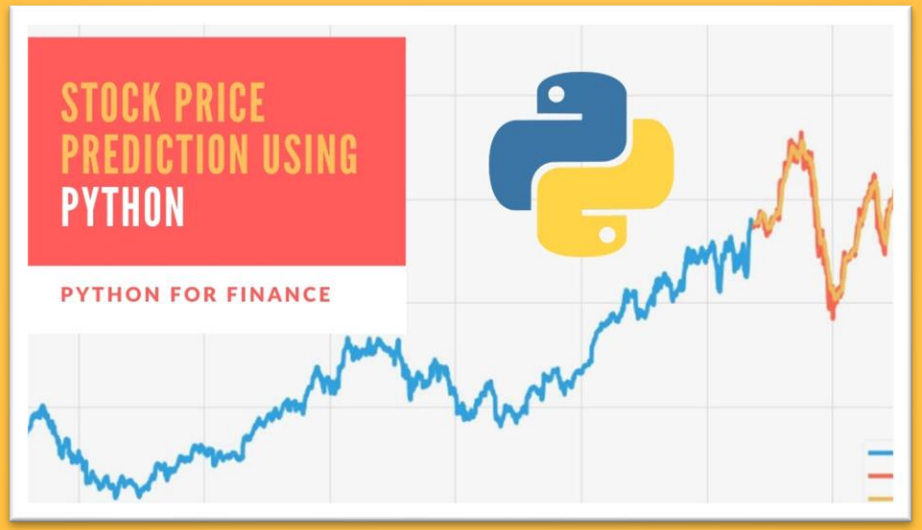
머신러닝으로 코스피 주식 가격 예측하기



CUAI 3기 Machine Learning Track A팀
공태웅(공공인재학부), 권예진(응용통계학과), 김원준(국제물류학과)

CONTENTS

01
서론

01 서론

‘주식’ 에 대한 관심 증대

지금 한국은 ‘주식’열풍

코스파가 사상 최고치를 갱신하고 있는 현 상황에, 개미와 기 투자자 모두 너나 할 거 없이 “주식 투자”에 뛰어 들고 있다. 특히, 개미 투자자가 역대 최고치의 공매도율을 보이며 수익을 보고 있는 상황이다.
또한, 애, 어른 할 거 없이 전 세대에 걸쳐 주식에 대한 관심과 참여가 증가하였고, 2020년의 끝과 2021년의 초반의 주 키워드는 ‘주식’이라고 볼 수 있다. 이에 발맞추어 ‘주식’ 관련 주제를 선정한 것은 당연하다.

주제 선정

**과거 기업 지표 데이터를 학습 피쳐로 사용해,
머신러닝 기법으로 특정 기업의 곧 다가올 주가를 예측**

이미 수많은 기업에서 애널리스트와 전문 투자자들이 하고 있는 일이지만,
현재 배운 머신러닝을 사용해 주식을 예측해본다는 것에 의의를 둠.

02
본론

02 본론

데이터 수집 및 변수 확인

수집한 데이터 요약

1. 기업의 영업 실적과 부채가 담긴 데이터
(분기별 영업/순이익률, 부채비율 등)

2. 투자 심리가 담긴 데이터
(공매도 비율, 외국인 보유 주식 수, 투자의견)

3. 기업의 규모
(시가총액, 매매단위, 상장주식수 등)

* 현재 기준으로 코스피 상장사는 총 800사이지만, 각종 우선주와 데이터가 이례적으로 결여된 기업들을 제외하여 총 766개 (약 96%) 의 레이블을 구축

* 근 5개 분기 (19년 3,4분기, 20년 1,2,3분기) 별로 펼쳐 수집

네이버 금융 (www.finance.naver.com) 사이트에는 기업 실적 데이터가 DART로부터 데이터를 받아 대리 제공하는데, 이를 크롤러를 사용해 데이터셋을 구축하였다. 이에 공매도 수량과 같이 DART에서 다루지 않는 데이터는 한국거래소(KRX)의 공매도 데이터를 수집해 병행하여 데이터셋을 보충

데이터 전처리

```
def fill_na(few_na_df):
    df = few_na_df.copy(deep=True)
    t = df.isna().sum() > 0
    t = t[t == True]
    na_columns = t.index.to_list()
    na_columns.append('업종코드')
    grouped_by_industry_code = df.groupby(['업종코드'], dropna=False).mean()

    for na_col in na_columns:
        # na가 아닌 항목들은 index_error를 일으켜서 해당 에러를 무시하면서 계속 돌도록
        try:
            na_mean_df = grouped_by_industry_code[na_col] # 업종코드와 na_col의 평균 값을 가지고 있는 DF
            comp_with_industry_codes = df[df[na_col].isna()]['업종코드']  # index: company_name, value: company_industry_code

            for company, code in zip(comp_with_industry_codes.index, comp_with_industry_codes.values):

                mean_value = na_mean_df.loc[code]

                # 평균 값이 nan이 아닐 때 산업부문 별 평균 값 넣고
                if not np.isnan(mean_value):
                    df.loc[company, na_col] = mean_value

                # 평균 값이 nan이면 해당 칼럼 중간값 넣기
                else:
                    df.loc[company, na_col] = df[na_col].median()
        except:
            pass
    return df
```

Nan 값은 해당 산업군의 평균 값으로 채우되,
해당 산업군의 평균 값도 Nan일 경우, 해당 칼럼의 중간 값으로 채움.

```
df = pd.read_csv('./data/data_filled.csv', encoding='utf-8-sig')

df = df.drop(['시가총액', '기업명'], axis=1)

encoder = LabelEncoder()
df['업종코드'] = encoder.fit_transform(df['업종코드'])
df['업종코드'] = df['업종코드'].astype(np.object)
```

미래산업으로 부각되는 특정 업종은 기업의 평가에도 긍정적으로 평가된다는 점을 착안하여 업종 데이터(업종코드)를 원 핫 인코딩 처리하여 학습에 사용

데이터 모델링

회귀 관련 모델은 피쳐 간 다중공선성 문제가 우려되기에,
랜덤 포레스트 모델(트리모델)을 사용해 모델링을 진행하였습니다.

```
def treeModel(self, max_depth, min_samples_split, min_samples_leaf, n_estimators):
    params = {}
    params['max_depth'] = int(round(max_depth))
    params['min_samples_split'] = min_samples_split
    params['min_samples_leaf'] = min_samples_leaf
    params['n_estimators'] = int(n_estimators)
    params['random_state'] = 32
    params['class_weight'] = 'balanced'

    rf = RandomForestClassifier(**params)
    rf.fit(self.X_train, self.y_train)

    pred = rf.predict(self.X_valid)
    result = mean_squared_error(self.y_valid, pred)
    return -result
```

파라미터 튜닝

베이지안 옵티마이저를 사용해 파라미터를 튜닝
기반 지식이 없는 상황에서 튜닝에 대한 명확한 기준이 없었기 때문에,
튜닝 작업을 최대한 자동화하고자 RFTuner 클래스를 정의하여 사용하였다.

```python
def train_model(self, pbounds, n_iter=15, verbose=2, init_points=5):

    '''pbounds = {
        'max_depth': (5, 30),
        'min_samples_split': (0.1, 0.5),
        'min_samples_leaf': (0.1, 0.25),
        'n_estimators': (10, 100)
    }'''

    # pbouns에 해당하는 지표들의 인자를 넥하여 가장 좋은 결과를 낸 값을 선정
    brf = BayesianOptimization(f=self.treeModel, pbounds=pbounds, verbose=verbose)
    brf.maximize(init_points=init_points, n_iter=n_iter)
    print('best_target_value:', brf.max['target'])

    # 위의 과정에서 구한 최적의 패러미터 대입
    params = {}
    params['max_depth'] = int(round(brf.max['params']['max_depth']))
    params['min_samples_split'] = brf.max['params']['min_samples_split']
    params['min_samples_leaf'] = brf.max['params']['min_samples_leaf']
    params['n_estimators'] = int(brf.max['params']['n_estimators'])
    params['random_state'] = 32
    params['class_weight'] = 'balanced'
```

RMSE를 낮추는 방향으로 파라미터 튜닝을 진행했고,
55회의 이터레이션 끝에 파라미터를 결정할 수 있었다.

In [5]:
```
tuner.train_model(pbounds, n_iter=50)
```

iter	target	max_depth	min_sa...	min_sa...	n_esti...
1	-9.05e+09	15.62	0.4424	0.4648	2e+03
2	-6.24e+09	28.42	0.1912	0.1115	2.001e+0
3	-1.117e+1	6.899	0.2287	0.1516	2e+03
4	-9.05e+09	27.01	0.4444	0.3777	2.001e+0
5	-9.05e+09	26.66	0.3912	0.2295	2e+03
6	-6.094e+0	29.43	0.1	0.1	2.001e+0
7	-9.05e+09	29.07	0.49	0.4828	2e+03
8	-7.931e+0	28.82	0.3207	0.4624	2.001e+0
9	-5.499e+0	28.45	0.1597	0.1415	2.001e+0
10	-5.732e+0	28.51	0.1261	0.249	2e+03
11	-6.354e+0	28.3	0.1048	0.4635	2.001e+0
12	-1.454e+1	23.82	0.2942	0.332	2.001e+0
13	-5.617e+0	29.0	0.1069	0.1259	2.001e+0
14	-6.094e+0	30.0	0.1	0.1	2e+03
15	-5.617e+0	13.09	0.1069	0.2134	2.001e+0
16	-6.106e+0	6.443	0.164	0.2304	2e+03
17	-5.826e+0	5.994	0.1002	0.3082	2.001e+0
18	-5.193e+0	29.46	0.1257	0.1214	2.001e+0
19	-1.752e+1	29.64	0.2665	0.2391	2.001e+0
20	-1.472e+1	28.21	0.3054	0.3985	2e+03
21	-1.499e+1	28.62	0.1499	0.2645	2.001e+0
22	-6.375e+0	26.67	0.1724	0.4377	2.001e+0
23	-5.81e+09	28.96	0.1702	0.4562	2.001e+0
24	-1.249e+1	27.04	0.2461	0.2137	2e+03
25	-1.461e+1	22.04	0.2984	0.3562	2e+03
26	-1.275e+1	22.02	0.2413	0.1274	2e+03
27	-1.749e+1	28.38	0.2495	0.1419	2.001e+0
28	-9.05e+09	19.92	0.4465	0.1564	2e+03
29	-5.292e+0	21.55	0.1317	0.3312	2e+03
30	-8.688e+0	20.78	0.3296	0.1393	2e+03
31	-6.162e+0	29.4	0.1657	0.1436	2.001e+0
32	-1.249e+1	7.677	0.2467	0.1496	2.001e+0
33	-8.688e+0	23.52	0.3297	0.3528	2.001e+0
34	-6.499e+0	28.48	0.1701	0.1226	2.001e+0
35	-9.05e+09	16.8	0.3432	0.3196	2e+03
36	-9.05e+09	14.52	0.47	0.1111	2e+03
37	-5.303e+0	21.91	0.1028	0.3975	2e+03
38	-8.597e+0	21.81	0.1	0.3849	2e+03
39	-1.485e+1	28.48	0.1457	0.1844	2.001e+0
40	-5.372e+0	13.07	0.1407	0.1653	2.001e+0
41	-5.392e+0	13.05	0.1397	0.2452	2.001e+0
42	-1.395e+1	13.08	0.2043	0.1895	2.001e+0
43	-9.05e+09	6.005	0.4079	0.1646	2.001e+0
44	-9.262e+0	21.92	0.1	0.4353	2e+03
45	-5.467e+0	29.42	0.1232	0.1485	2.001e+0
46	-6.216e+0	21.57	0.1817	0.3174	2e+03
47	-9.05e+09	14.58	0.3606	0.2832	2.001e+0
48	-9.05e+09	21.86	0.4158	0.3862	2.001e+0
49	-1.332e+1	16.06	0.1982	0.3068	2e+03
50	-8.285e+0	6.063	0.1072	0.3055	2.001e+0
51	-5.392e+0	29.46	0.14	0.2008	2.001e+0
52	-9.05e+09	6.988	0.4137	0.4221	2.001e+0
53	-6.128e+0	29.37	0.1628	0.2037	2.001e+0
54	-7.617e+0	21.6	0.3267	0.3485	2e+03
55	-9.05e+09	23.49	0.3656	0.3637	2.001e+0

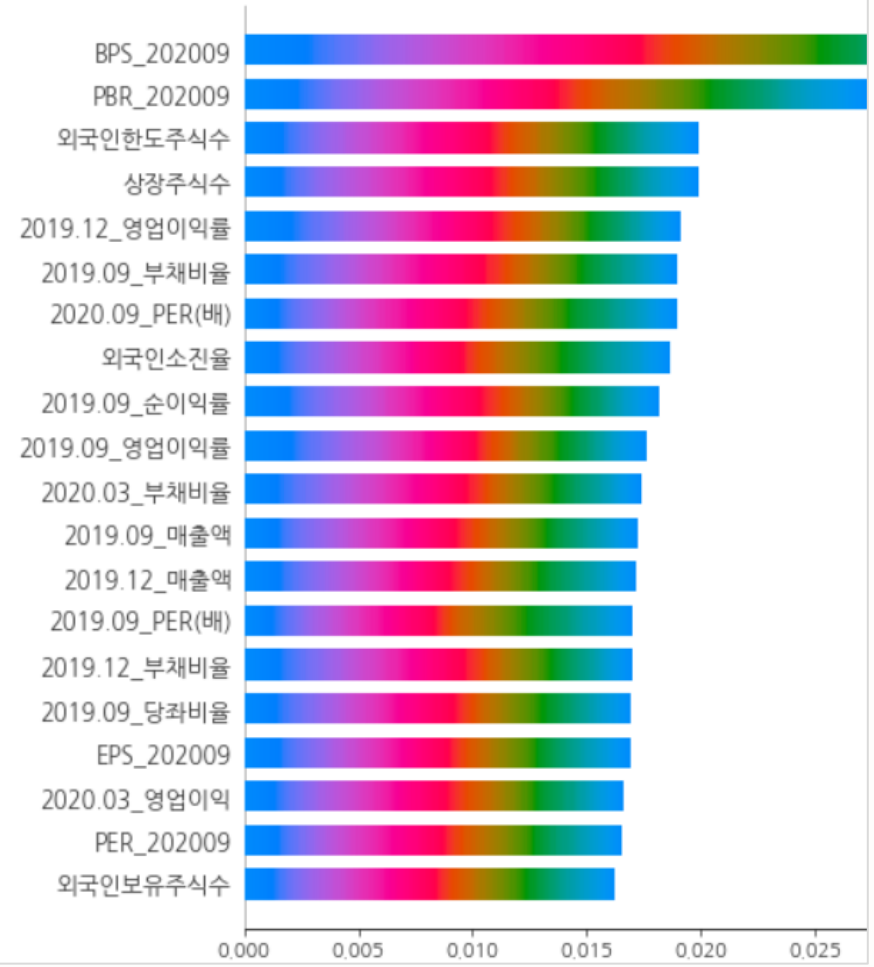
```
best_target_value: -5192506649.928572
Training with best parameters ...
parameter file rf_5192506649.9286.params'' is saved successfully

RMSE:  72059.04974344702
```

(+) 피쳐 중요도

```
exp, shap_val = tuner.get_shap_values()
```

```
Creating legend with loc="best" can be slow with large amounts of data.
```



전분기 BPS(주당 순자산 가치)와 전분기 PBR(주가순자산비율)이 다음 분기 주가 예측에 중요한 피처임을 알 수 있다.

⇒ BPS와 PBR 모두 주식과 자산에 대한 피처이다.

⇒ 즉, 한 주당 기업의 자산이 어떻게 가치 변하는지가, 차후 주가 예측에 중요하다는 것을 알 수 있다.

03
결론

03 결론

결론 및 정리

- 주가에 영향을 미치는 데이터를 크롤링한 다음,
랜덤 포레스트 모델을 사용해 학습시키면서, 미래의 주가를 예측해보았다.

- 베이지안 옵티마이저를 사용해, 의미 있는 파라미터를 뽑아 내었고,
실제 주식 투자 활동에 활용하는 과정에서 많은 인사이트를 얻어낼 수 있었다.

03 결론

개선할 점

- 회귀 관련 모델을 사용해, 좀 더 통계학적으로 예측 모델로서 가치 있는 모델을 만들지 못한 점.
- 이미 여러 투자사들이 사용하고 있는 여러 주가 관련 예측 모델과 비교할 때 낮은 수준의 모델일 수 밖에 없다고 생각하나, 시계열로 주식을 보는 관점이 아니라, 머신러닝 트리 기반의 지도학습으로 주식을 예측해 보았다는 관점이 이번 컨퍼런스 연구의 중요한 점이다.
- 시계열로 주식을 바라보지 않았다는 점에서, 시간이라는 주식에서 가장 중요한 피처를 사용하지 않았다. 시간이라는 중요한 피쳐를 고려한 예측 모델(Ex.ARIMA 모델) 을 테스트로라도 시도해보아 트리 기반 모델과의 성능 비교도 해보았으면 좋았을 것 같다.