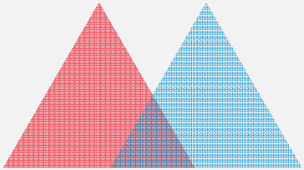
아파트 실거래가 예측 분석



8조 이상민 윤대관 윤태영 김태인 이상우

Contents

01 train 데이터 추가 탐색

- 대치동 13년도 이전 데이터
- 대치동 인접지역 데이터

train 데이터 추가 탐색

[제공 데이터]

- 대치동의 14년도~21년도 아파트 실거래가 데이터

- 거래 연월, 거래일자, 실 거래 가격, 층수 등

[추가 데이터]

- 대치동의 06년도~13년도 아파트 실거래가 데이터
- 대치동과 인접한 4개 동(역삼동, 개포동, 일원동, 도곡동)의 06년도~23년도 아파트 실거래가 데이터
- 위도/경도 데이터 수집

(국도교통부 실거래가 공개시스템(http://rtdown.molit.go.kr/)에서 다운로드 후 전처리, 하나의 파일로 병합)

train 데이터 추가 탐색

```python
# 대치동 데이터 추가
for i in range(6, 14):
    # 파일 경로 설정
    file_path = f'./대치동데이터/대치{i:02}.csv'
    new_file_path_cleaned = f"./대치동데이터/train_dachi{i:02}.csv"

    # 데이터 읽어오기
    data = pd.read_csv(file_path, skiprows=15,encoding='cp949')
    train = pd.read_csv('./train.csv')

    # 열 이름 변경
    data = data.rename(columns={
        '시군구': 'sigungu',
        '번지': 'jibun',
        '단지명': 'apt_name',
        '전용면적(㎡)': 'exclusive_use_area',
        '계약년월': 'transaction_year_month',
        '계약일': 'transaction_day',
        '거래금액(만원)': 'transaction_real_price',
        '층': 'floor',
        '건축년도': 'year_of_completion'
    })

    # 필요없는 열 제거
    columns_to_drop = ['본번', '부번', '도로명', '해제사유발생일', '등기신청일자', '거래유형', '중개사소재지']
    data = data.drop(columns=[col for col in columns_to_drop if col in data.columns])

    # 'id' 열 추가
    data['id'] = 'TRAIN_' + data.index.astype(str)

    # 열 순서 변경
    data = data[train.columns]

    # Save the cleaned DataFrame to a new CSV file
    data.to_csv(new_file_path_cleaned, index=False)
```

```python
combined_data = pd.DataFrame([])

for i in range(6, 14):
    print(f'{i}년도 불러옴...')
    data = pd.read_csv(f'./대치동데이터/train_dachi{i:02}.csv')
    data['transaction_real_price'] = data['transaction_real_price'].str.replace(',', '').astype(int)
    combined_data = pd.concat([combined_data, data])

# Reset the index
data = pd.read_csv('train.csv')
combined_data = pd.concat([combined_data, data])
combined_data.reset_index(drop=True, inplace=True)

combined_data.to_csv("train_dachi.csv", index=False)
```

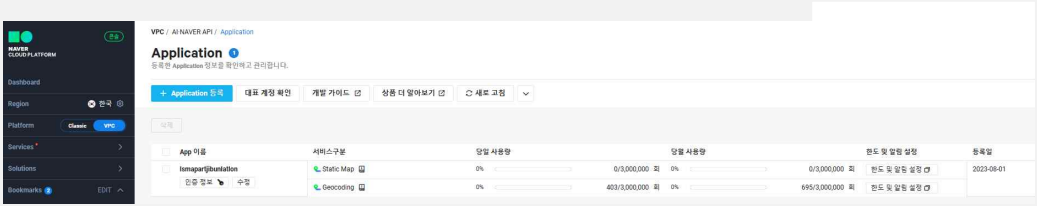
```
6년도 불러옴...
7년도 불러옴...
8년도 불러옴...
9년도 불러옴...
10년도 불러옴...
11년도 불러옴...
12년도 불러옴...
13년도 불러옴...
```

- 대치06.csv
- 대치07.csv
- 대치08.csv
- 대치09.csv
- 대치10.csv
- 대치11.csv
- 대치12.csv
- 대치13.csv

02 위도/경도 데이터 수집

▪지번으로부터 위도/경도 데이터 수집

위도/경도 데이터 수집



- 지번이 **key**, 좌표가 **value**인 딕셔너리 생성
- 네이버 Geocoding API를 활용, 각각의 동에 대해 지번으로부터 위도/경도 데이터를 확보, train과 test 데이터에 추가하여 새 feature 생성

위도/경도 데이터 수집

```
train_dic={}
for adress in train['jibun'].value_counts().index:
    juso = "서울특별시 강남구 대치동 " + str(adress)
    add_urlenc = parse.quote(juso) # URL Encoding
    url = api_url + add_urlenc
    request = Request(url)
    request.add_header('X-NCP-APIGW-API-KEY-ID', Client_ID)
    request.add_header('X-NCP-APIGW-API-KEY', Client_Secret)

    try:
        response = urlopen(request)

    except HTTPError as e:
        print('HTTP Error')
        latitude, longitude = None, None

    else:
        rescode = response.getcode()

        if rescode == 200:
            response_body = response.read().decode('utf-8')

            if response_body['addresses'] == []:
                print('No result')
            else:
                latitude = response_body['addresses'][0]['y']
                longitude = response_body['addresses'][0]['x']
                print('Success')
        else:
            print(f'Response error, rescode:{rescode}')
            latitude, longitude = None, None
    train_dic[adress]=[latitude, longitude]
```

```
train['lat']=0
train['lon']=0
test['lat']=0
test['lon']=0
```

```
for i in range(len(train)):
  key=train['jibun'][i]
  train['lat'][i]=float(train_dic[key][0])
  train['lon'][i]=float(train_dic[key][1])
```

```
for i in range(len(test)):
  key=test['jibun'][i]
  test['lat'][i]=float(test_dic[key][0])
  test['lon'][i]=float(test_dic[key][1])
```

```
train.to_csv('train_xy.csv', index=False)
test.to_csv('test_xy.csv', index=False)
```

```
#이런 식으로 지번과 위경도가 대응
train_dic
```

```
{'757': ['37.4975233', '127.0500650'],
 '706-20': ['37.5019590', '127.0442572'],
 '755-4': ['37.4987845', '127.0498940'],
 '836': ['37.4919817', '127.0330105'],
 '709': ['37.5025026', '127.0463890'],
 '720-25': ['37.5014592', '127.0427008'],
 '716': ['37.5024446', '127.0493831'],
 '762': ['37.4955829', '127.0469173'],
 '763': ['37.4970057', '127.0480859'],
 '722': ['37.5005747', '127.0423208'],
 '606-18': ['37.5064893', '127.0338076'],
 '761-10': ['37.4962760', '127.0463985'],
 '754-1': ['37.4977619', '127.0467993'],
 '713-11': ['37.4993252', '127.0458108'],
 '824-25': ['37.4975914', '127.0311371'],
 '711': ['37.5013296', '127.0437126'],
 '712': ['37.5016391', '127.0455446'],
 '766-8': ['37.4961420', '127.0504294'],
 '716-1': ['37.5017736', '127.0496857'],
 '795-41': ['37.4927617', '127.0384403'],
 '708-31': ['37.5031250', '127.0483602'],
 '759-1': ['37.4971787', '127.0476774'],
 '771-4': ['37.4973162', '127.0428446'],
 '835-67': ['37.4920882', '127.0349001'],
 '747-5': ['37.4957106', '127.0378712'],
 '727-11': ['37.4973395', '127.0394001'],
 '754': ['37.4985802', '127.0464368'],
 '710': ['37.5014748', '127.0449372'],
 '789-29': ['37.4941653', '127.0356756'],
 '787-4': ['37.4936176', '127.0426849'],
```

03 다양한 feature 생성

- 금리
- 마트/백화점, 학교, 역과의 거리 추가

금리 데이터 구하기

3.2 금리 데이터

```
def preprocess_tran_date(x):
    if type(x) == int:
        if x < 10:
            return '0'+str(x)
        else:
            return str(x)
    else:
        return x

train['transaction_day'] = train['transaction_day'].apply(preprocess_tran_date)
train['transaction_date'] = train['transaction_year_month'].astype(int).astype(str) + train['transaction_day'].astype(str)
train['transaction_date'] = pd.to_datetime(train['transaction_date'])
train = train.sort_values('transaction_day').reset_index(drop=True)
```

```
def make_date(row):
    month_day = row['월일'].replace('월 ', '-')
    month_day = month_day.replace('일', '')
    date = str(row['연도'])+ '-' + month_day
    return date

interest_rate['날짜'] = interest_rate.apply(lambda x: make_date(x), axis=1)
interest_rate['날짜'] = pd.to_datetime(interest_rate['날짜'])
```

```
from tqdm import tqdm
from datetime import datetime

for idx, row in tqdm(train.iterrows(), total = train.shape[0]):
    date = row['transaction_date']
    rate = interest_rate[interest_rate['날짜'] <= date].iloc[0]['금리']
    train.loc[idx, 'interest_rate'] = rate
```

```
100%|██████████| 4909it/4909it [00:50<00:00, 964.93it/s]
```

```
test['transaction_day'] = test['transaction_day'].apply(preprocess_tran_date)
test['transaction_date'] = test['transaction_year_month'].astype(int).astype(str) + test['transaction_day'].astype(str)
test['transaction_date'] = pd.to_datetime(test['transaction_date'])
test = test.sort_values('transaction_day').reset_index(drop=True)
```

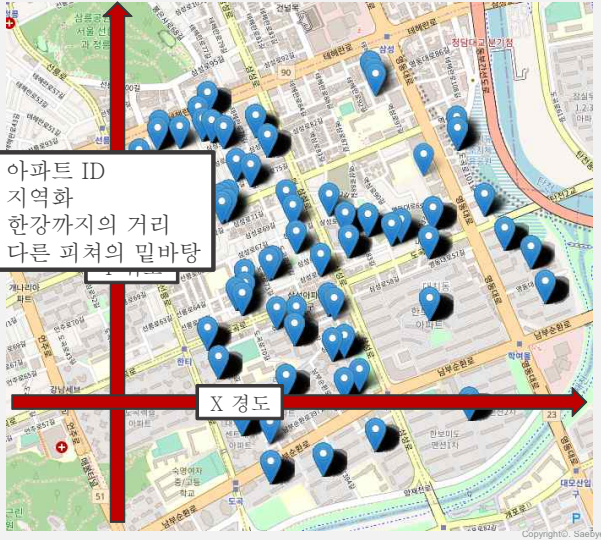
```
for idx, row in tqdm(test.iterrows(), total = test.shape[0]):
    date = row['transaction_date']
    rate = interest_rate[interest_rate['날짜'] <= date].iloc[0]['금리']
    test.loc[idx, 'interest_rate'] = rate
```

```
100%|██████████| 196/196 [00:00<00:00, 936.32it/s]
```

```
train['transaction_day']=train['transaction_day'].map(lambda x: int(x))
train = train.drop('transaction_date',axis  = 1)
test = test.drop('transaction_date',axis  = 1)
```

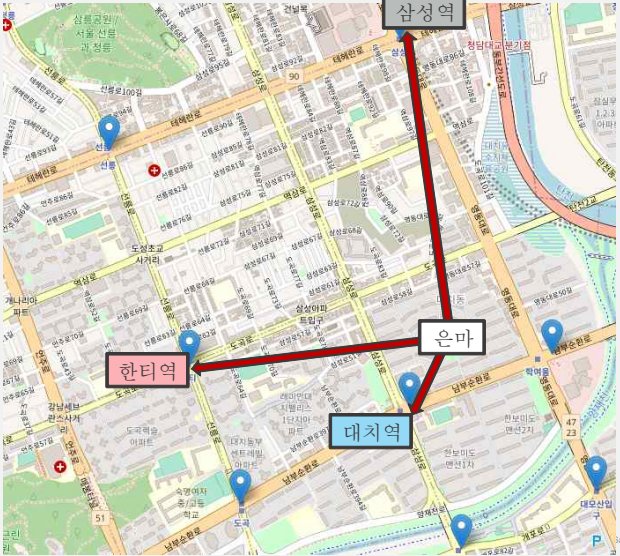
네이버 API를 활용한 위도/경도 구하기

	apt_name	lat	lon
0	한보미도맨션2	37.493383	127.067333
7	은마	37.497414	127.065309
12	대치아이파크	37.495037	127.054580
19	대치삼성	37.496388	127.058412
26	대치동우정에쉐르1	37.504547	127.054224
29	대치삼성	37.496388	127.058412
38	대치현대	37.500835	127.060986
40	대치현대	37.500835	127.060986
69	한보미도맨션1	37.493383	127.067333
72	한보미도맨션2	37.493383	127.067333
73	한보미도맨션1	37.493383	127.067333
76	한보미도맨션1	37.493383	127.067333
79	풍림아이원4차(1007-2)	37.504256	127.066691
107	은마	37.497414	127.065309
109	은마	37.497414	127.065309
113	선경3차	37.494174	127.060947
115	선경3차	37.494174	127.060947
117	은마	37.497414	127.065309
118	은마	37.497414	127.065309
119	은마	37.497414	127.065309



역세권 구하기 위해 역까지의 최단거리 구하기

	apt_name	subway_dist
0	한보미도맨션2	0.35317
1	경남1	0.48669
2	개포주공7단지	0.29533
3	래미안도곡카운티	0.24226
4	경남1	0.48669
5	디에이치아너힐즈	0.56410
6	디에이치아너힐즈	0.56410
7	은마	0.34450
8	경남1	0.48669
9	도곡택슬	0.36359
10	SK허브프리모	0.13076
11	대우디오빌	0.27655
12	대치아이파크	0.20120
13	경남2차	0.48669
14	도곡우성아파트	0.50180
15	도곡택슬	0.36359
16	뉴현대파크빌	1.38373
17	타워팰리스2	0.14967
18	타워팰리스2	0.14967
19	대치삼성	0.48897



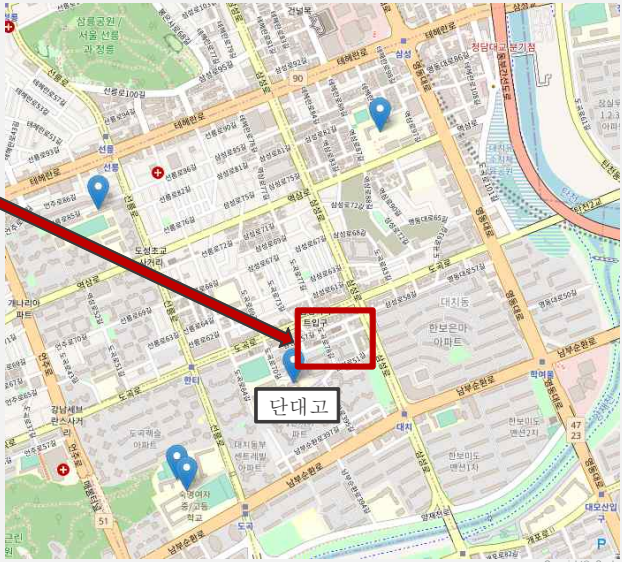
학군구하기

학교 거리 구하기

```
[13] school_lation = pd.read_csv('전국초중등학교위치표준데이터.csv', encoding = 'cp949')

[14] school=school_lation[school_lation['교육지원청명']=='서울특별시강남서초교육지원청']

[ ] train['school_dist']=0
    test['school_dist']=0

[ ] for i in range(len(train)):
      distance=[]
      for j in range(len(school)):
        y1=train.iloc[i]['lat']
        x1=train.iloc[i]['lon']
        x2=school.iloc[j]['위도']
        y2=school.iloc[j]['경도']
        start = (x1,y1)
        goal = (x2,y2)
        r=haversine(start, goal)
        distance.append(r)
      train['school_dist'][i]=round(np.min(distance),5)
```

강남 8학군으로 범위를 제한

	apt_name	hakgun_dist
0	한보미도맨션2	0.73808
7	은마	0.69455
12	대치아이파크	0.28854
19	대치삼성	0.10284
26	대치동우정에쉐르1	0.58016
29	대치삼성	0.10284
38	대치현대	0.50320
40	대치현대	0.50320
69	한보미도맨션1	0.73808
72	한보미도맨션2	0.73808
73	한보미도맨션1	0.73808
76	한보미도맨션1	0.73808
79	풍림아이원4차(1007-2)	0.42998
107	은마	0.69455
109	은마	0.69455
113	선경3차	0.32315
115	선경3차	0.32315
117	은마	0.69455
118	은마	0.69455
119	은마	0.69455



대형마트 백화점까지의 거리

▾ 마트 백화점 거리 구하기

```
[21] mart = pd.read_csv('서울시 강남구 대규모점포 인허가 정보.csv', encoding = 'cp949')
```

```
[22] mart2=mart[mart['영업상태명']=='영업/정상']
     mart3=mart2[mart2['점포구분명']!='준대규모점포']
     mart3=mart3[mart2['업태구분명']!='그 밖의 대규모점포']
     mart4=mart3[mart3['좌표정보(X)'].notnull()]
     mart5=mart4.drop_duplicates(['좌표정보(X)'])
     mart6=mart5[['좌표정보(X)','좌표정보(Y)']]
```

```
[23] def project_array(coord, p1_type, p2_type):
         """
         좌표계 변환 함수
         - coord: x, y 좌표 정보가 담긴 NumPy Array
         - p1_type: 입력 좌표계 정보 ex) epsg:5179
         - p2_type: 출력 좌표계 정보 ex) epsg:4326
         """
         p1 = pyproj.Proj(init=p1_type)
         p2 = pyproj.Proj(init=p2_type)
         fx, fy = pyproj.transform(p1, p2, coord[:, 0], coord[:, 1])
         return np.dstack([fx, fy])[0]
```

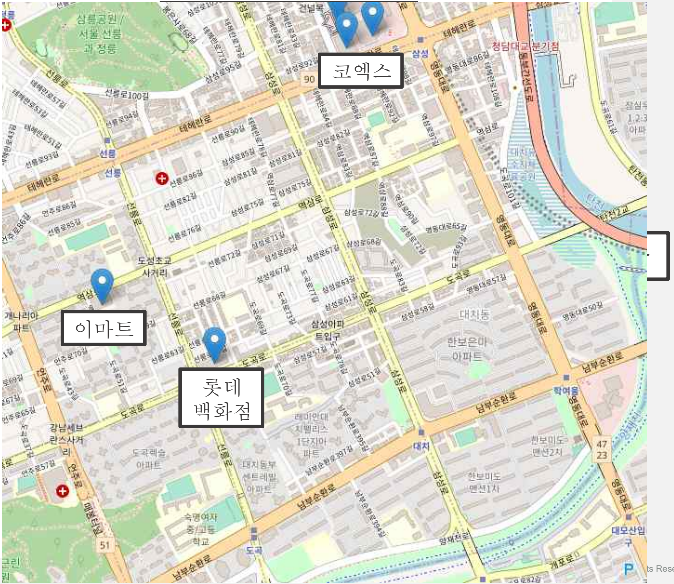
```
[24] coord = np.array(mart6)
```

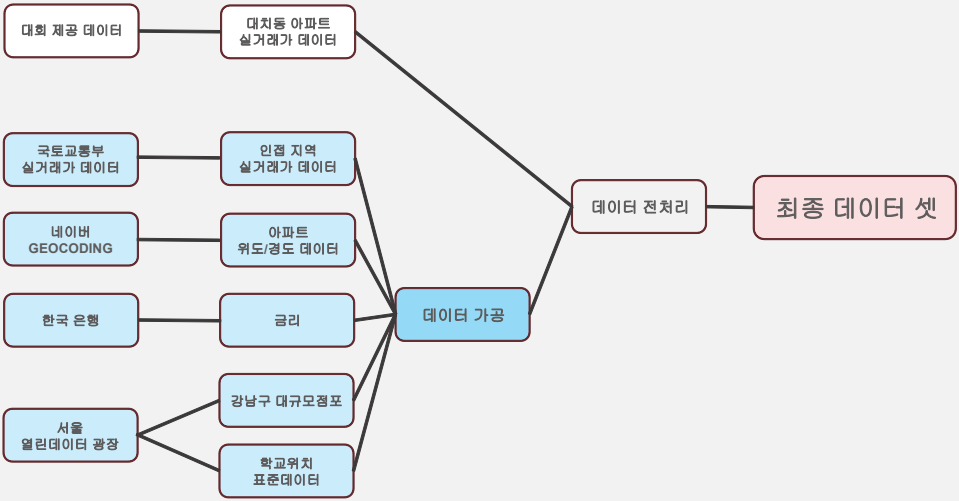
```
[25] p1_type = "epsg:2097"
     p2_type = "epsg:4326"

     # project_array() 함수 실행
     result = project_array(coord, p1_type, p2_type)
```

```
[26] mart6['lat'] = result[:, 1]
     mart6['lon'] = result[:, 0]
     mart6['사업장명']=mart5['사업장명']
     mart6['lat']=mart6['lat'].map(lambda x: x-0.0001)
     mart6['lon']=mart6['lon'].map(lambda x: x+0.0029)
     mart_dict=mart6[['사업장명','lat','lon']]
```

```
[ ] train['mart_dict']=0
    test['mart_dict']=0
```

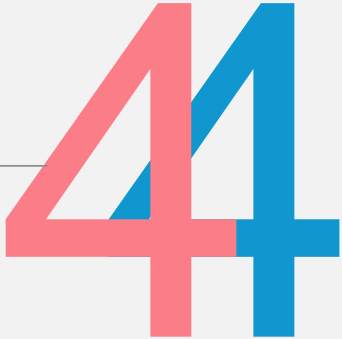
```
[ ] for i in range(len(train)):
      distance=[]
      for j in range(len(mart_dict)):
        x1=train.iloc[i]['lat']
        y1=train.iloc[i]['lon']
        x2=mart_dict.iloc[j]['lat']
        y2=mart_dict.iloc[j]['lon']
        start = (x1,y1)
        goal = (x2,y2)
        r=haversine(start, goal)
        distance.append(r)
      train['mart_dict'][i]=round(np.min(distance),5)
```





04 모델링

- XGBoost
- Feature 제거

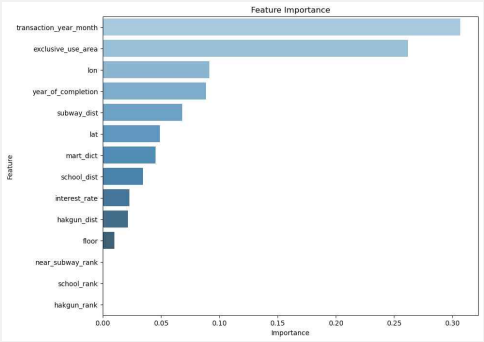


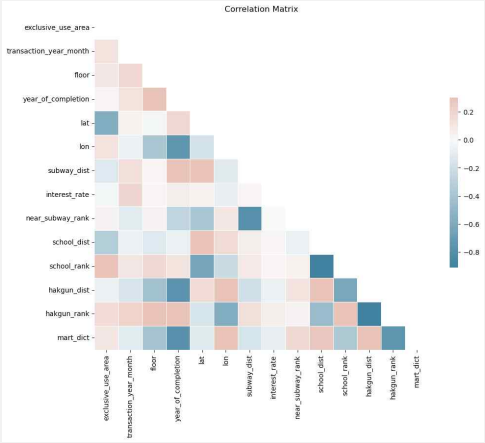
변수 선택

5 트레인 feature 추가버전에서 인덱스, 시군구, 지번, 아파트이름, 거래일 column 제거

```
import pandas as pd

# CSV 파일로부터 데이터를 불러옴
df = pd.read_csv('./train_feature6.csv')

# 데이터프레임의 첫 5행을 표시함
df.head()

# 지정된 열을 제거함
df = df.drop(['index', 'sigungu', 'jibun', 'apt_name', 'transaction_day'], axis=1)

# 수정된 데이터프레임을 CSV 파일로 저장함
df.to_csv('./train_feature6.csv', index=False)

# CSV 파일로부터 테스트 데이터를 불러옴
df_test = pd.read_csv('./test_feature6.csv')

# 테스트 데이터에서 지정된 열을 제거함
df_test = df_test.drop(['sigungu', 'jibun', 'apt_name', 'transaction_day'], axis=1)

# 수정된 데이터프레임을 CSV 파일로 저장함
df_test.to_csv('./test_feature6.csv', index=False)
```

시각화





XGBoost

XGBoost는 Extreme Gradient Boosting의 약자
Boosting 기법을 이용하여 구현한 알고리즘은 Gradient Boost 가 대표적
XGBoost는 알고리즘을 병렬 학습이 지원되도록 구현한 라이브러리
Regression, Classification 문제를 모두 지원하며, 성능과 자원 효율이 좋아서, 인기 있게 사용되는 알고리즘

장점

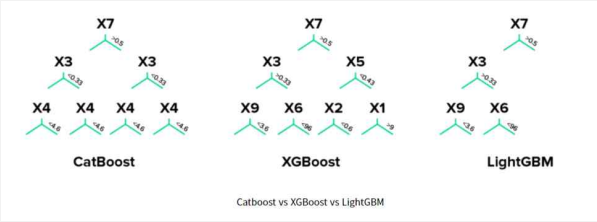
- GBM 대비 빠른 수행시간
- 병렬 처리로 학습, 분류 속도가 빠름
- 과적합 규제(Regularization)
- 표준 GBM 경우 과적합 규제기능이 없으나, XGBoost는 자체에 과적합 규제 기능으로 강한 내구성
- 분류와 회귀영역에서 뛰어난 예측 성능 발휘
- 즉, CART(Classification and regression tree) 앙상블 모델을 사용
- Early Stopping(조기 종료) 기능이 있음
- 다양한 옵션을 제공하며 Customizing이 용이

XGBoost 선택이유

LightGBM은 DFS(깊이 우선 탐색)처럼 트리를 우선적으로 깊게 형성하는 방식

XGBoost는 BFS(너비 우선 탐색)처럼 우선적으로 넓게 트리를 형성

CatBoost는 이렇게 Feature를 모두 동일하게 대칭적인 트리 구조를 형성



XGBoost 와 CatBoost와 차이점이 없는 것처럼 보이지만
트리가 나누어지는 Feature들의 대칭 여부에 따라 차이
(데이터와 피쳐를 많이 넣다보니 과적합이 문제되었고, 데이터를 추가함으로 분류가 잘 되어야 했기 때문에 XGBoost를 선택)
다른 모델로 학습 시 정확도가 좋지 않았음

XGBoost

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
import xgboost as xgb

# 훈련 데이터를 불러옵니다
df_train = pd.read_csv('./train_feature6.csv')

# 데이터를 입력 특징(X)과 목표 변수(y)로 나눕니다
X = df_train.drop(['id', 'transaction_real_price'], axis=1)
y = df_train['transaction_real_price']

# 데이터를 훈련 세트와 검증 세트로 나눕니다
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# XGBoost Regressor 모델을 생성하고 훈련 데이터에 적합시킵니다
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma=0, subsample=0.75, colsample_bytree=1,
                             max_depth=7, random_state=42)
xgb_model.fit(X_train, y_train)

# 훈련된 모델을 사용해 검증 세트의 실제 거래 가격을 예측합니다
y_val_pred_xgb = xgb_model.predict(X_val)

# 예측의 평균 절대 오차(MAE)를 계산합니다
mae_xgb = mean_absolute_error(y_val, y_val_pred_xgb)

# 테스트 데이터를 불러오고 그 특징을 훈련 데이터와 일치하도록 재정렬합니다
df_test = pd.read_csv('./test_feature6.csv')
df_test_reordered = df_test[['id'] + X.columns.tolist()]
```

```python
# 훈련된 모델을 사용해 테스트 데이터의 실제 거래 가격을 예측합니다
y_test_pred_xgb = xgb_model.predict(df_test_reordered.drop(['id'], axis=1))

# 예측된 가격을 테스트 데이터프레임에 추가합니다
df_test['transaction_real_price'] = y_test_pred_xgb

# 'id'로 데이터프레임을 정렬합니다
df_test_sorted = df_test.sort_values(by='id')

# 예측된 가격을 CSV 파일로 저장합니다
df_final_xgb = df_test_sorted[['id', 'transaction_real_price']]
df_final_xgb.to_csv('./submission0802xgb.csv', index=False)
```

모델 평가

881579	submission0802xgb.csv edit	2023-08-02 20:44:44	12151.7915897438 -	◉
881016	predictions.csv 일원, 개포, 대치 3개 동의 데이터를 train 데이터로 활용, 시군구 column LabelEncoding, RandomForestRegressor edit	2023-08-02 10:16:04	21063.3656410256 -	○
880802	1234.csv edit	2023-08-02 00:18:21	21335.1619615385 -	○
880774	subcat (1).csv edit	2023-08-01 23:59:33	21422.1555139064 -	○
880708	subdnn.csv edit	2023-08-01 22:56:15	46766.9602564103 -	○
880626	submissiontarget.csv edit	2023-08-01 21:16:13	29986.0012820513 -	○
879574	predictions.csv edit	2023-07-31 22:24:24	20991.8667948718 -	○
879340	submission3.csv edit	2023-07-31 15:13:28	27740.8381666667 -	○
878903	Lasso+GridSearch.csv edit	2023-07-30 21:05:20	113191.4425627445 -	○

후기

- ✓ 데이터 전처리의 중요성을 느낌
- ✓ 코드 정리
- ✓ 팀원들과의 소통

감사합니다