Google Developer Student Clubs

Week6 Presentation

Kaggle House Prices - Advanced Regression Techniques
Code review & Performance improvements

GDSC Hanyang ML/DL : Basic, Jaeseung Lee

Index

- Code Review
  - Data preprocessing
  - Applying ML
- Performance Improvement
  - 시도한 방법들
  - 결과

Code Review

Data Preprocessing

1. dropna
2. Using statistical graphs
    1. Remove not helpful columns
    2. Remove outliers
3. Factorize categorical columns
4. Using correlation of columns
5. Checking VIF

Code Review

Dropna

- 결측값을 가지는 column들을 제거하는 방식
- 결측값이 있고, 결측값을 채워넣을 방법이 없을 때 분석 신뢰성 확보를 위해 사용

```
# 결측값을 가지는 것들의 합을 나타냄
df.isnull().sum()
```

✓ 0.0s

```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                 ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

```
# 결측값이 하나라도 있는(any) feature(axis=1)들 없앰
df = df.dropna(axis=1, how='any')
```

✓ 0.0s

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

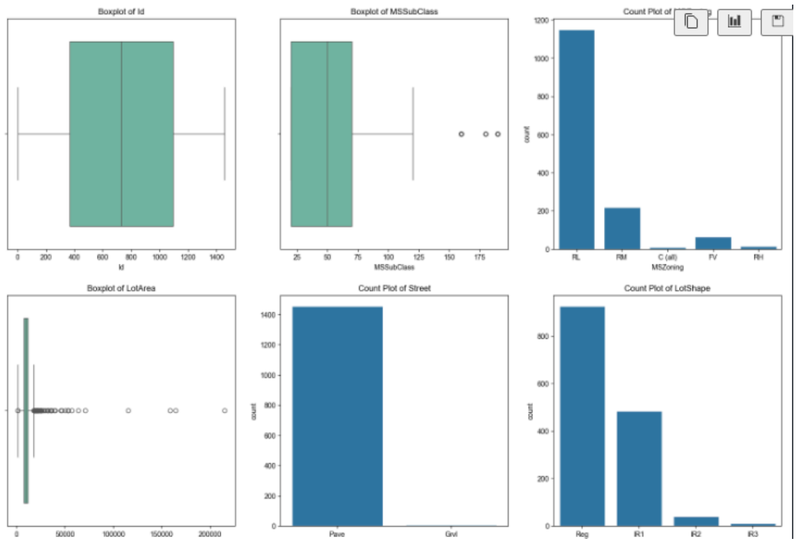
✓ 0.0s

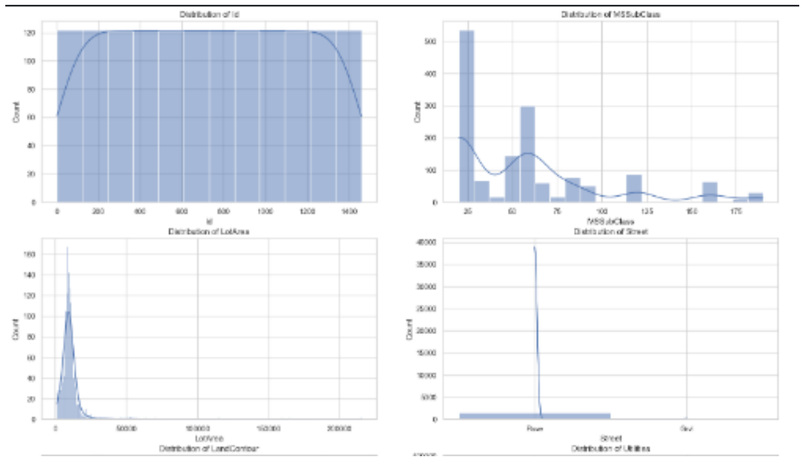
```
Id              0
MSSubClass      0
MSZoning        0
LotArea         0
Street          0
                ..
MoSold          0
YrSold          0
SaleType        0
SaleCondition   0
SalePrice       0
Length: 62, dtype: int64
```

Code Review

Using statistical graphs & method

- 데이터들의 Boxplot이나, Distribution 그래프를 활용
- 분석에 별로 도움이 되지 않는 column 제거
  - Distribution이 과도하게 치중되어 있는 경우
  - Outlier들이 많은 경우
- Outlier들을 제거
  - quantile(분위수)를 통해 범위 벗어나는 데이터 제거





Code Review

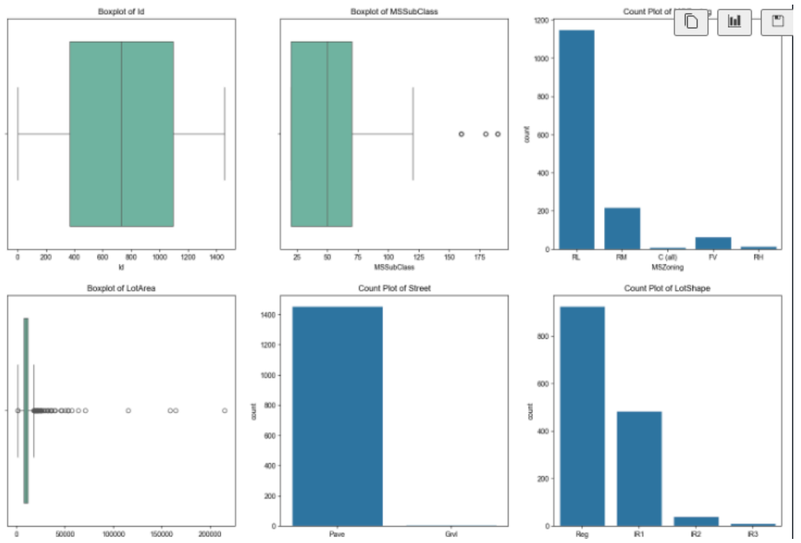
Using statistical graphs & method

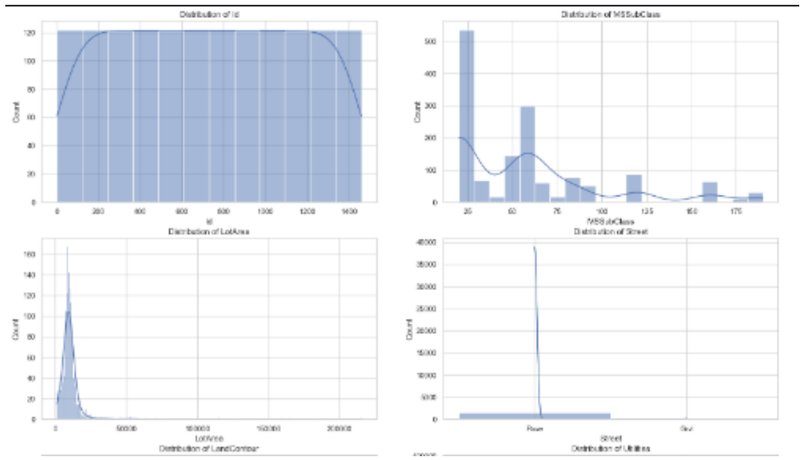
- 데이터들의 Boxplot이나, Distribution 그래프를 활용
- 분석에 별로 도움이 되지 않는 column 제거

```
    # 도움안되는거 잘라버리기, inplace = True : 원본 또한 삭제
    ptr_df.drop(column_are_not_helpful_in_prediction, axis=1, inplace=True)
```

- Outlier들을 제거
  - quantile(분위수)를 통해 범위 벗어나는 데이터 제거

```
# outlier 제거하는 과정
def remove_outliers(df, columns, threshold=1.5):
    for column in columns:
        # quantile : 분위수, 0.25 => 25% 를 구해줌.
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold * IQR
        upper_bound = Q3 + threshold * IQR
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df
```





Code Review

Factorize categorical columns

- 현재 데이터에는 int64 type만 있는게 아니라 object도 있다.
- 이 object 안에는 값과 범주가 있는데 이 중 값만 뽑아 데이터를 학습에 용이하게 변경
- pd.factorize() 함수가 그 역할을 함.

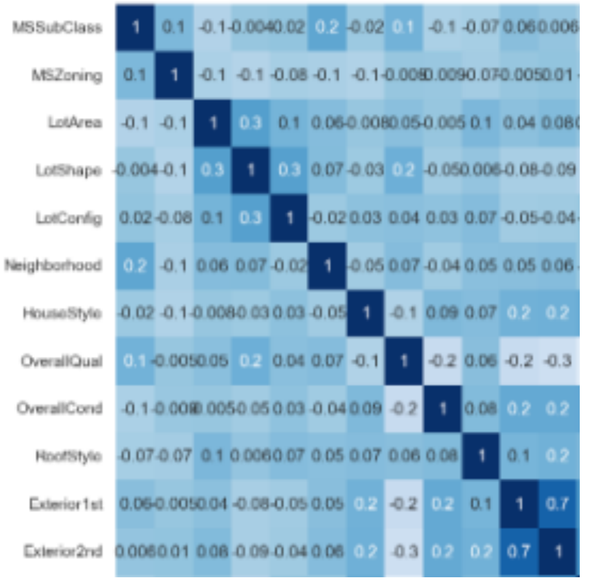
```
# 이 함수는 object에 있는 값을 꺼내 분석에 활용하기 좋은 int64값으로 변경하는 과정
def factorize_categorical_columns(column):
    if column.dtype == 'object':
        # 객체가 있으면 그 객체의 값과 범주를 나눠줌
        column_encoded, _ = pd.factorize(column)
        return column_encoded
    return column

# Apply factorize only to categorical columns
df_encoded = removed_outlier.apply(factorize_categorical_columns)
```

Code Review

Using correlation of columns

- Correlation analysis
  - 상관분석은 두 변수관의 관계의 강도를 상관계수로 표현하는 방식
  - 피어슨 상관계수
    - -1 : 음적 선형관계, 0 : 선형관계 X, +1 양적 선형관계
- 이를 통해 우리의 예측 target인 Saleprice와 다른 column들을 분석



Code Review

Using correlation of columns

```python
# Find highly correlated features
highly_correlated_features = set()
for i in range(len(cor.columns)):
    for j in range(i):
        if abs(cor.iloc[i, j]) > 0.5:
            colname_i = cor.columns[i]
            colname_j = cor.columns[j]
            highly_correlated_features.add(colname_i)
            highly_correlated_features.add(colname_j)

# Convert the set of highly correlated features to a list
highly_correlated_features_list = list(highly_correlated_features)
```

Code Review
Checking VIF

- Variance Inflation Factor (분산팽창인수)
  - 데이터들이 다중공산성이 있는지 확인함
    - 다중공산성 : 독립변수들 간에 강한 상관관계가 나타나는 문제
    - 이는 데이터 분석 시 부정적인 영향 미침

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF : 분산팽창인수, 독립변수가 다중공산성이 있는지 확인하는 것.
vif=pd.DataFrame()
vif['VIF']=[variance_inflation_factor(sel_df,i) for i in range(sel_df.shape[1])]
vif['features']=sel_df.columns
```

✓ 0.1s

Code Review

Applying ML

- Sklearn을 통해 Linear Regression을 진행함

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

model = LinearRegression()
# 모델 학습시에는 train dataset 사용
model.fit(X_train, y_train)
```

```
# testset은 모델 예측할 때 사용됨.
test_pred=model.predict(X_test)
end_pred=pd.DataFrame(test_pred,index=df_test.index)
end_pred.columns=['SalePrice']
end_pred.to_csv('submission_base.csv',sep=',')
end_pred.head()
```

Code Review

Performance Improvement

- scikit-learn이 익숙하지 않아 수정하고 사용하는데 난항이 있었음.
- 먼저 데이터 전처리는 이미 Base code에 잘 되어있어 건드리지 않음.
- 비교적 간단하게 기술된 Applying ML 부분이 수정도 쉽고 추가할 부분이 많기에 수정
- 사용했던 방식
  - 기본 LinearRegression이 아닌 다른 Model 사용
    - Ridge(L2-norm), ElasticNet(L1+L2), RandomForestRegressor, …
  - KFold-Cross Validation, Grid Search 사용
- 출처 : *scikit-learn 정리* (kdhangelic, 2022.02.07)

Code Review

Performance Improvement

- 사용했던 모델 정리 및 방식
  - Lasso, Ridge, **ElasticNet** : 모두 L1, L2 norm regularization과 관련된 모델
  - **RandomForestRegressor** : RandomForest (decision tree 기반)를 통한 reg
  - **GradientBosstingRegressor** : Gradient Boosting Algorithm 사용
  - **XGBRegressor** : XGBoost라는 외부 모델을 사용.
  - **StackingRegressor** : 앞서 나온 모델들을 Stacking 하여 더 높은 성능을 기대 가능

Code Review

Performance Improvement

```python
from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import ElasticNet

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor

from xgboost import XGBRegressor

poly_pipeline = make_pipeline(
    PolynomialFeatures(degree=2, include_bias= False), StandardScaler(), ElasticNet(alpha=0.1, l1_ratio=0.2)
)
rfr = RandomForestRegressor()
gbr = GradientBoostingRegressor(random_state=42, learning_rate =0.01)
xgb = XGBRegressor(random_state=42, learning_rate=0.01, n_estimators=1000, subsample = 0.8, max_depth=3)
```

```python
from sklearn.linear_model import LinearRegression

X_train=X
y_train=y

model_basic = LinearRegression(n_jobs=-1)

model_basic.fit(X_train, y_train)
print(model_basic.score(X_train, y_train))

stack_models = [
    ('elasticnet', poly_pipeline),
    ('randomforest', rfr),
    ('gbr', gbr)
]

#model = poly_pipeline # score = 0.89
#model = rfr # score = 0.978
#model = rfr_gs # score = 0.95
#model = gbr # score = 0.78
#model = xgb # score = 0.936
#model = StackingRegressor(stack_models, final_estimator=xgb, n_jobs=-1) #0.92
#model.fit(X_train, y_train)
#model.score(X_train, y_train)
```

Code Review

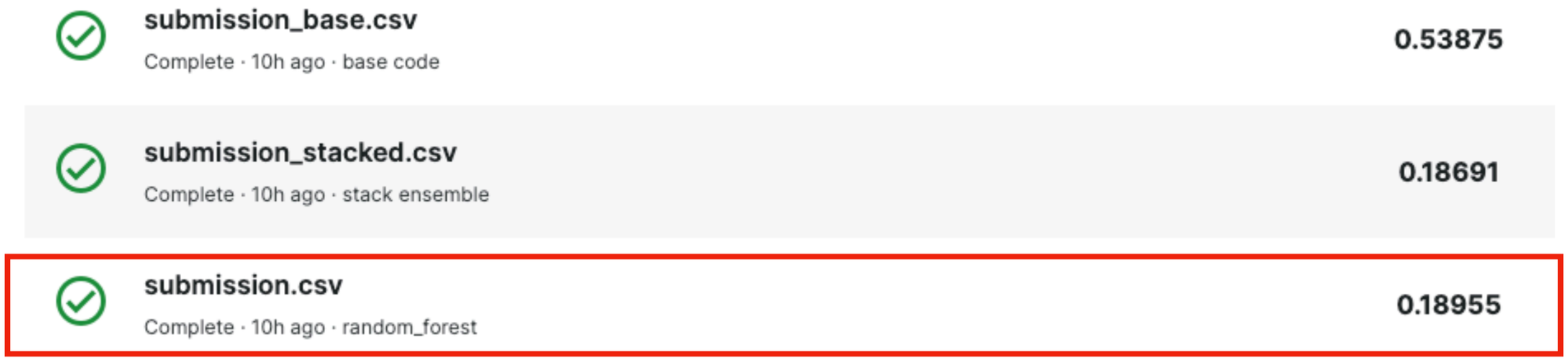
Performance Improvement

- 각 모델 성능에 따른 model.score(x,y) 결과
  - Lasso, Ridge, **ElasticNet** : 0.89
  - **RandomForestRegressor** : **0.978**
  - **GradientBosstingRegressor :** 0.78
  - **XGBRegressor** : 0.936
  - **StackingRegressor** :0.92

Code Review

Performance Improvement

- 그러나,,, 기존의 코드보다 훨씬 결과가 떨어지게 나옴 => train score만 이용했기 때문



Code Review

Performance Improvement

- 그래서 다른 방법 추가로 사용
  - GridSearchCV : 최적의 성능을 내는 hyperparameter를 찾아주는 API
  - 가장 결과가 좋았던 RandomForestRegressor에 적용 => 0.95 나옴,, => stacked

```
from sklearn.model_selection import GridSearchCV
params = {
    'n_estimators': [500, 1000],
    'max_depth': [7, 8],
    'max_features': [0.8, 0.9,],
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(RandomForestRegressor(), params, cv=3, n_jobs=-1, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
rfr_gs = RandomForestRegressor(max_depth = 8, max_features = 0.8, n_estimators = 100)
```

Code Review

Performance Improvement

- 시도한 방식
  - KFold Cross Validation : train/test dataset을 나누는 방식, 교차검증을 진행함.
  - 코드는 남아있지 않지만 아래와 같은 코드로 진행 (출처 참고)

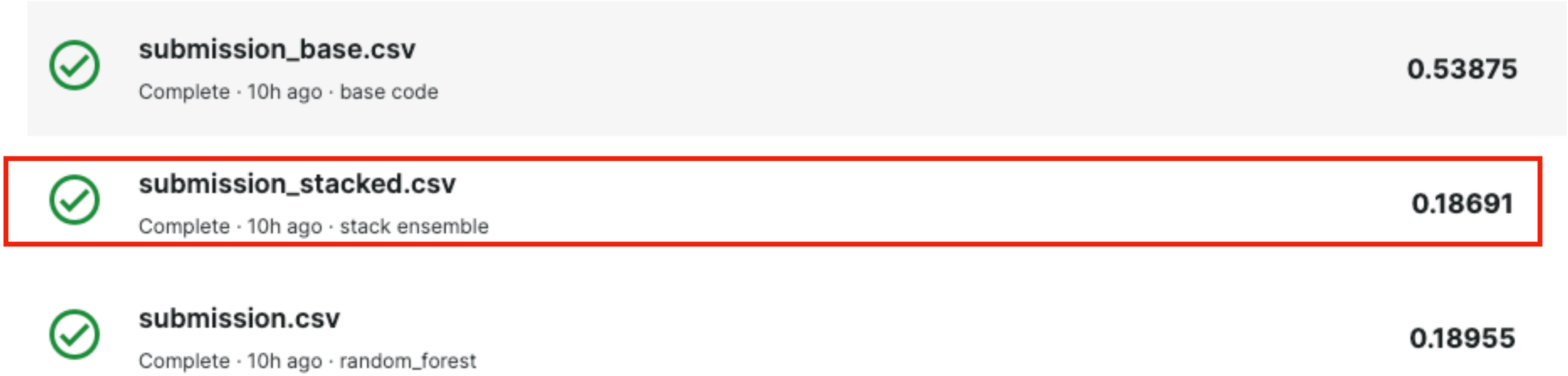
```
from sklearn.model_selection import KFold
n_splits = 5
kfold = KFold(n_splits=n_splits, random_state=42)
X = np.array(df.drop('MEDV', 1))
Y = np.array(df['MEDV'])
lgbm_fold = LGBMRegressor(random_state=42)

i = 1
total_error = 0
for train_index, test_index in kfold.split(X):
    x_train_fold, x_test_fold = X[train_index], X[test_index]
    y_train_fold, y_test_fold = Y[train_index], Y[test_index]
    lgbm_pred_fold = lgbm_fold.fit(x_train_fold, y_train_fold).predict(x_test_fold)
```

Code Review

Performance Improvement

- 이렇게 해도 그냥 LinearRegression() 보다 못한 결과를 보게 되었음.
- 시간이 없어 더 진행하지 못한 것이 아쉽지만, 데이터 전처리의 중요성과 괜히 건드리는게 오히려 역효과를 낼 수 있다는 것을 깨달았던 시간이 되었던 것 같음.



Google Developer Student Clubs

Thank you for listening

Kaggle House Prices - Advanced Regression Techniques
Code review & Performance improvements

GDSC Hanyang ML/DL : Basic, Jaeseung Lee