Machine Learning 2nd Term project

Regression – Food Delivery Time Prediction

2021741062 김현준

데이터 선정

분류 알고리즘때와 동일하게 Kaggle에서 데이터를 찾았음. (https://www.kaggle.com/). 분류 알고리즘 때 사용했던 데이터는 (https://www.kaggle.com/datasets/thedevastator/higher-education-predictors-of-student-retention) Columns이 34개였고, 삭제할 데이터가 없었기에 이로 인한 과적합이 일어났음. 그래서 feature_importances를 사용하여 예측에 사용되지 않는 부분을 선택한 후 열을 줄이는 방식으로 과적합을 예방함. 그랬기에 이번에 사용할 데이터 셋은 그 부분에 유의하여 적당한 Columns를 가진 데이터를 선정하였음.

이번에 사용하는 데이터 셋은 Food Delivery Time Prediction이라는 데이터로, 다양한 파라미터와 배달까지 걸리는 시간의 상관관계를 체크할 수 있는 데이터 셋임. Xlsx 형식으로 올라와 있었기 때문에 CSV로 저장하여 전처리를 진행하였음.

데이터 전처리

	ID	Delivery_person_ID	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery_location_latitude	Delivery_location_long
0	4607	INDORES13DEL02	37	4.9	22.745049	75.892471	22.765049	75.91
1	B379	BANGRES18DEL02	34	4.5	12.913041	77.683237	13.043041	77.81
2	5D6D	BANGRES19DEL01	23	4.4	12.914264	77.678400	12.924264	77.68
3	7A6A	COIMBRES13DEL02	38	4.7	11.003669	76.976494	11.053669	77.02
4	70A2	CHENRES12DEL01	32	4.6	12.972793	80.249982	13.012793	80.28
...	...	...	...	...	...	...	...	
45588	7C09	JAPRES04DEL01	30	4.8	26.902328	75.794257	26.912328	75.80
45589	D641	AGRRES16DEL01	21	4.6	0.000000	0.000000	0.070000	0.07
45590	4F8D	CHENRES08DEL03	30	4.9	13.022394	80.242439	13.052394	80.27
45591	5EEE	COIMBRES11DEL01	20	4.7	11.001753	76.986241	11.041753	77.02
45592	5FB2	RANCHIRES09DEL02	23	4.9	23.351058	85.325731	23.431058	85.40

사용하는 데이터는 총 45593 개의 rows 와 11 개의 columns 로 이루어져 있음. 11 개의 열은 ID, Delivery person ID, Delivery person Age, Delivery person Rating, Restaurant latitude, Restaurant

longitude, Delivery location latitude, Delivery location longitude, Type of order, Type of vehicle, Time taken(min)

이 Columns 중 Drop 할 Columns 가 있음. ID 와 Delivery person ID 는 음식의 배달 시간 예측 학습에 영향이 없는 부분으로 수업시간에 타이타닉 생존자 예측을 계산할 때, PassengerID (탑승자 데이터 일련 번호), Name(탑승자 이름), Ticket(티켓 번호)을 Drop 한 것과 같은 맥락임.

```
def drop_features(df):
    df.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
    return df
```

그 전에 데이터에 빈칸이 있는지 확인하기 위해 data.info() 사용하였고, 출력결과 데이터 결측치는 없는 것으로 확인되었음.

```
In [2]: print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           45593 non-null  object
 1   Delivery_person_ID           45593 non-null  object
 2   Delivery_person_Age          45593 non-null  int64
 3   Delivery_person_Ratings      45593 non-null  float64
 4   Restaurant_latitude          45593 non-null  float64
 5   Restaurant_longitude         45593 non-null  float64
 6   Delivery_location_latitude   45593 non-null  float64
 7   Delivery_location_longitude  45593 non-null  float64
 8   Type_of_order                45593 non-null  object
 9   Type_of_vehicle              45593 non-null  object
 10  Time_taken(min)              45593 non-null  int64
dtypes: float64(5), int64(2), object(4)
memory usage: 3.8+ MB
None
```

그 다음 데이터를 확인해보면 다음과 같이 위도 경도가 0 으로 나타나는 음식점이 있음을 확인할 수 있음. 위도와 경도가 모두 0 인 지점은 태평양이기 때문에 현실적으로 불가능하다고 생각하여 위도와 경도가 0 인 데이터를 삭제하였음. 그 결과 400 개 정도의 데이터가 정리되었음.

25	5	0	0	0.03	0.03	Snack	motorcycle	15
28	4.8	11.003681	76.975525	11.083681	77.055525	Drinks	motorcycle	36
31	4.7	21.173343	72.792731	21.233343	72.852731	Snack	motorcycle	16
24	4.7	10.96185	76.971082	11.05185	77.061082	Buffet	motorcycle	19
39	4.6	27.165108	78.015053	27.255108	78.105053	Snack	motorcycle	27
39	4.2	0	0	0.08	0.08	Snack	scooter	49
32	4.5	26.88842	75.800689	26.89842	75.810689	Buffet	scooter	20
36	4.7	0	0	0.06	0.06	Snack	motorcycle	30

```
data = data[(data['Restaurant_latitude'] != 0) & (data['Restaurant_longitude'] != 0)]
data
```

그 다음으로는 경도와 위도 데이터를 사용해서 의미가 있는 데이터로 바꾸는 작업을 진행하였음. 현재 주어진 위도 경도 데이터는 단순한 수치 데이터기 때문에 아무런 정보가 없는 상황임. 또한 위도 경도 데이터에 – 가 붙은 경우도 있기에 – 가 붙은게 맞는 상황인지 검증하고자 하였음.

```
from geopy.distance import geodesic

def calculate_distance(row):
    restaurant_coords = (row['Restaurant_latitude'], row['Restaurant_longitude'])
    delivery_coords = (row['Delivery_location_latitude'], row['Delivery_location_longitude'])
    return geodesic(restaurant_coords, delivery_coords).kilometers

data.loc[:,'Distance'] = data.apply(calculate_distance,axis=1)
data
```

다음과 같이 distance 를 구하였고, 그 distance 를 다시 걸린 시간으로 나누어 Speed 를 계산하였음. Speed 를 계산한 다음 각 운송수단 별로 평균 Speed 를 구하였는데, 구해진 평균 Speed 는 다음과 같음.

```
Type_of_vehicle
bicycle              11062.110627
electric_scooter       565.471390
motorcycle             190.060214
scooter                326.818548
Name: Speed_kmph, dtype: float64
```

말이 안 되는 데이터가 섞여 있기에 다음과 같이 평균 수치가 급등하였는데, 이와 같이 결측치는 아니지만 에러 데이터를 걸러내기 위해 Speed 가 1000km/h 보다 넘게 측정된 행을 제거하였음. 제거 후 새롭게 구한 평균 Speed 는 다음과 같음

```
Type_of_vehicle
bicycle             20.853279
electric_scooter    25.521418
motorcycle          22.501621
scooter             25.520499
Name: Speed_kmph, dtype: float64
```

경도, 위도 데이터와 거리, 속도, 시간 데이터가 모두 존재할 경우 각 데이터가 서로 연관이 되어 있기에 과적합이 유발될 가능성이 굉장히 큼. 그렇기에 경도, 위도, 속력데이터를 삭제하였음.

그 다음은 범주형 데이터를 학습에 사용할 수 있게 바꿔주는 과정이 필요한데, 분류 알고리즘과 다르게 중요한 라벨 인코딩은 사용할 수 없고 One-Hot 인코딩만 사용할 수 있음. 지난번 프로젝트 때는 pd.get_dummies(data)를 적용했을 때 True False 로 Output 이 나와서 이번에는 dtype 를 지정하여 출력하였음.

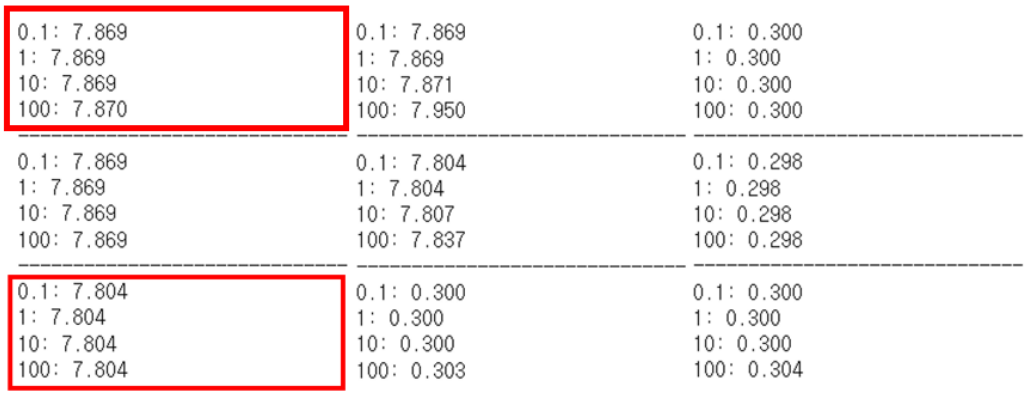
```
data = pd.get_dummies(data,dtype = int)
data
```

Type_of_order_Snack	Type_of_vehicle_bicycle	Type_of_vehicle_electric_scooter	Type_of_vehicle_motorcycle	Type_of_vehicle_scoo
1	0	0	1	
1	0	0	0	
0	0	0	1	
0	0	0	1	
1	0	0	0	
...	...	...	...	
0	0	0	1	
0	0	0	1	
0	0	0	0	
1	0	0	1	
1	0	0	0	

범주형 데이터에 대해서 전처리를 한 후에는 숫자형 데이터에 대해서 전처리를 진행 해야함. 숫자형 데이터에 대한 전처리는 정규화, 표준화, n 차 다항평가, 로그 변환 등이 있음. 가지고 있는 데이터로 어떤 전처리가 제일 적합할지 확인해보기 위해 evaluate 함수를 만들어 테스트를 진행하였음.

```python
def evaluate(X, y):
    alphas = [0.1, 1, 10, 100]
    for alpha in alphas:
        ridge = Ridge(alpha=alpha)
        neg_mse_scores = cross_val_score(ridge, X, y, scoring='neg_mean_squared_error', cv=5)
        rmse_scores = np.sqrt(-1 * neg_mse_scores)
        avg_rmse = np.mean(rmse_scores)
        print(f'{alpha}: {avg_rmse:.3f}')
```

Evaluate 함수는 Ridge 회귀모델을 사용해서 X, y 에 대해 교차검증을 진행한 후 다양한 alpha 값에 대한 평균 Rmse 를 계산함. Rmse 는 Root Mean Squared Error 로 예측 값과 실 제값 간의 차이를 측정하는 성능지표임.



성능지표를 통해 검사한 결과 1 보다 작은 값들은 과적합을 예상하여 사용하지 않았음. 7.804 의 수치는 표준화에 2 차 다항평가를 진행한 결과로 제일 좋은 성적을 보여주었음. 하지만 학습에 사용한 데이터는 7.869 의 점수를 보여준 표준화만을 사용하여 진행하였음.

2 차 다항평가는 선형 데이터가 비선형의 성질을 가질 수 있도록 변환하는 역할을 함. 그렇기에 비선형 알고리즘에는 부적합한 전 처리 방식일 수 있음. 그랬기에 선형/비선형 알고리즘을 모두 사용하는 이번 프로젝트와는 맞지 않은 전 처리 방식이라고 생각하였음. 또한 다항평가를 진행할 경우 열의 개수가 77 개로 너무 늘어나기 때문에 표준화만을 진행하여 혹시 모를 과적합을 방지하고자 하였음.

분류 텀 프로젝트에서는 모든 전처리가 완료된 후 Train, Test 셋을 분류하였는데, 이후 공부를 통해 이전과 같은 방식으로 Train, Test 를 나누면 Train 의 전 처리 과정에서 Test 의 데이터가 영향을 끼칠 수 있음을 알게 되었음.

그랬기에 이번 프로젝트에서는 표준화를 진행하기 전 train_test_split 를 활용하여 Train, Test 를 분류하고 이후 표준화를 진행하여 혹시 모를 오차를 사전에 방지하고자 하였음.

또한 총 데이터의 행이 41500 개인데, 총 데이터를 40000 개 이상으로 설정할 경우 학습시간이 무한정 길어지는 상황이 발생하여, 행의 개수를 조절하여 적절한 시간으로 학습을 완료할 수 있도록 하였음.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures

sampled_data = data.sample(n=30000, random_state=42) # 30000개만 사용
sampled_data.reset_index(drop=True, inplace=True) # Index 0부터 재배치
sampled_data
```

```
X = sampled_data.drop('Time_taken(min)', axis=1)
y = sampled_data['Time_taken(min)']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train= scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

전처리가 완료된 X_train 은 다음과 같음.

	0	1	2	3	4	5	6	7	8	9	10
0	-0.094976	0.215259	-1.205055	-0.569716	1.735522	-0.580750	-0.582739	-0.031639	-0.298541	0.846241	-0.708897
1	0.608116	0.827152	-1.463121	-0.569716	-0.576196	-0.580750	1.716034	-0.031639	3.349621	-1.181696	-0.708897
2	-0.798068	0.215259	-0.346863	-0.569716	-0.576196	1.721911	-0.582739	-0.031639	-0.298541	0.846241	-0.708897
3	1.486982	0.827152	-0.114533	-0.569716	-0.576196	-0.580750	1.716034	-0.031639	-0.298541	0.846241	-0.708897
4	1.486982	-2.538260	1.188705	-0.569716	-0.576196	1.721911	-0.582739	-0.031639	-0.298541	0.846241	-0.708897
...	...	...	...	...	...	...	...	...	...	...	...
23995	-1.325388	0.521205	-1.466136	-0.569716	1.735522	-0.580750	-0.582739	-0.031639	-0.298541	0.846241	-0.708897
23996	-0.446522	0.827152	1.864062	-0.569716	-0.576196	-0.580750	1.716034	-0.031639	-0.298541	-1.181696	1.410641
23997	-0.798068	1.133098	-0.351801	-0.569716	-0.576196	-0.580750	1.716034	-0.031639	-0.298541	0.846241	-0.708897
23998	-1.325388	0.215259	-0.925474	1.755260	-0.576196	-0.580750	-0.582739	-0.031639	-0.298541	0.846241	-0.708897
23999	0.783889	0.521205	-1.459177	-0.569716	1.735522	-0.580750	-0.582739	-0.031639	-0.298541	0.846241	-0.708897

데이터 학습

이번 학습에 사용하는 알고리즘은 다음과 같음

선형회귀 (Linear Regression, Lasso, Ridge, Elastic Net)

비선형회귀 (Decision Tree, Random Forest, SVR, XGB, GBM, ADABOOST)

지난번 텀 프로젝트에서는 따로 코드에 함수를 만들지 않고, 일일이 코드를 작성하여 코드의 길이가 길고, 가독성이 좋지 않았음. 그랬기에 이번에는 많이 쓰는 코드들을 함수화 시켜 코드를 간략하게 작성하였음.

이번에 사용하는 함수는 총 4 가지로 y_test, y_pred 를 바탕으로 plot 을 작성해주는 시각화 함수, 과대적합, 과소적합을 판단하기 위한 학습곡선을 그리는 함수, 입력받은 모델을 바탕으로 학습을 한 후 평가지표 MSE, MAE, R2 를 표시하는 평가함수, 그리고 각 모델의 파라미터 범위를 입력받으면 최적 하이퍼 파라미터를 계산한 후 최적 하이퍼 파라미터 값으로 재학습하여 모델을 평가하는 함수를 사용하였음.

시각화 함수

```
import matplotlib.pyplot as plt

def plot_draw(y_true, y_pred, title):
    plt.scatter(y_true, y_pred, s=1)
    z = np.polyfit(y_true, y_pred, 1)
    p = np.poly1d(z)
    trendline_x = np.linspace(min(y_true), max(y_true), 100)
    plt.plot(trendline_x, p(trendline_x), "r--")  # 추세선
    plt.plot(trendline_x, trendline_x, "g--")  # y = x 선
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.title(title)
    plt.show()
```

시각화 함수는 다음과 이루어져 있으며, y_true 값을 x 축, y_pred 값을 y 축으로 하여 그래프를 그리는 함수임. 그리고 각 점들의 추세선과 y = x 그래프를 같이 그리게 하였음. y = x 그래프를 같이 그리는 이유는, 예측률이 높을수록 y_true 와 y_pred 의 차이가 감소할 것이기에 추세선이 y = x 처럼 나타날 것이라고 예상하였기 때문임.

처음에는 y_test 에 대한 출력만 하려고 하였으나, y_train 데이터에 대한 그래프도 출력함으로써, train 부분이 과적합되고 있는 상황인지 판단하고자 하였음.

```
    # 훈련 및 테스트 데이터에 대한 예측 결과 시각화
    plot_draw(y_train, train_pred, 'Train Set')
    plot_draw(y_test, y_pred, 'Test Set')
```

학습곡선 그리는 함수

```
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=None, train_sizes=np.linspace(.1, 1.0, 100), scoring='neg_mea
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("RMSE (Lower is better)")

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=-1, train_sizes=train_sizes, scoring=scoring)

    train_scores_mean = np.sqrt(-np.mean(train_scores, axis=1)) # train rmse
    test_scores_mean = np.sqrt(-np.mean(test_scores, axis=1))   # test rmse

    plt.grid()
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", markersize=1, label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", markersize=1, label="Cross-validation score")
    plt.legend(loc="best")
    return plt
```

학습곡선은 Learning_curve 를 사용해서 나타낼 수 있으며, 주어진 모델에 따라 데이터 셋을 조절하며 테스트한 후 train, test rmse 를 제시하는 그래프임. rmse 말고 다른 요소들을 사용할 수 있지만, 본 학습에서는 rmse 를 사용하였음. 학습곡선을 그리는 이유는 trian, test size 가 일정한 비율로 잘 분리되었는지, 학습을 하면서 과대적합이나 과소적합이 일어나고 있는지 확인할 수 있기 때문임.

```
    # Learing Curve 그리기
    plot_learning_curve(best_model, "Learning Curve", X, y, cv=5)
```

모델에 따른 학습성능 체크 함수

```
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def evaluate_model(model, X_train, X_test, y_train, y_test): # 기본적으로는 test size를 0.2로 설정, 바꾸고 싶으면 test_size = 0.3 식으로 스

    # 모델 학습
    model.fit(X_train, y_train)

    # TRAIN 데이터 예측 및 평가
    train_pred = model.predict(X_train)
    train_mse = mean_squared_error(y_train, train_pred)
    train_mae = mean_absolute_error(y_train, train_pred)
    train_r2 = r2_score(y_train, train_pred)

    print("------Train Indicator-----")
    print(f"MSE: {train_mse}")
    print(f"MAE: {train_mae}")
    print(f"R² : {train_r2}")

    # TEST 데이터 예측 및 평가
    y_pred = model.predict(X_test)
    test_mse = mean_squared_error(y_test, y_pred)
    test_mae = mean_absolute_error(y_test, y_pred)
    test_r2 = r2_score(y_test, y_pred)

    print("------Test Indicator-----")
    print(f"MSE: {test_mse}")
    print(f"MAE: {test_mae}")
    print(f"R² : {test_r2}")

    # 테스트 세트에서 모델의 성능 확인을 위해 몇 가지 데이터를 무작위로 선택하여 예측 및 실제 라벨 표시
    random_idx = random.sample(range(len(y_test)), 5)
    random_data = X_test[random_idx]
    random_answer = y_test[random_idx]
    print("------Prediction Data-----")
    predictions = model.predict(random_data)
    rounded_predictions = np.round(predictions, 2)
    print("Predictions:₩t", rounded_predictions)          # 예측
    print("Labels:₩t₩t", list(random_answer))             # 실제

    # 훈련 및 테스트 데이터에 대한 예측 결과 시각화
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plot_draw(y_train, train_pred, 'Train Set')
    plt.subplot(1, 2, 2)
    plot_draw(y_test, y_pred, 'Test Set')

    plt.show()
```

학습성능 체크 함수 evluate_model 은 다음과 같이 진행됨. Train, Test 데이터에 대해 MSE MAE R2 지표를 출력함. 이후 5 개의 랜덤한 인덱스를 선택하여, 인덱스를 바탕으로 예측값과 정답을 출력함. 이 과정을 통해 MSE MAE R2 에 대한 수치보다 더 확실하게 예측이 잘 되고 있는 상황인지 파악할 수 있음. 이후 Train, Test 데이터에 대해 시각화 함수를 활용하여 그래프를 그림.

각 모델에 맞는 최적 하이퍼 파라미터 계산, 하이퍼 파라미터 값으로 다시 예측

```
from sklearn.model_selection import GridSearchCV

def evaluate_model_with_optimization(model, params, X_train, X_test, y_train, y_test):

    # 최적의 파라미터 탐색
    grid_search = GridSearchCV(model, params, cv=5, scoring='neg_mean_squared_error',n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # 최적의 파라미터 출력
    print("Best Parameters:", grid_search.best_params_)

    # 최적의 파라미터로 모델 재학습
    best_model = grid_search.best_estimator_
    best_model.fit(X_train, y_train)

    # 훈련 데이터 예측 및 평가
    train_predictions = best_model.predict(X_train)
    train_mse = mean_squared_error(y_train, train_predictions)
    train_mae = mean_absolute_error(y_train, train_predictions)
    train_r2 = r2_score(y_train, train_predictions)

    print("------Train Accuracy with Best Parameters-----")
    print(f"MSE: {train_mse}")
    print(f"MAE: {train_mae}")
    print(f"R² : {train_r2}")

    # 테스트 데이터 예측 및 평가
    y_pred = best_model.predict(X_test)
    test_mse = mean_squared_error(y_test, y_pred)
    test_mae = mean_absolute_error(y_test, y_pred)
    test_r2 = r2_score(y_test, y_pred)

    print("------Test Accuracy with Best Parameters-----")
    print(f"MSE: {test_mse}")
    print(f"MAE: {test_mae}")
    print(f"R² : {test_r2}")

    # 테스트 세트에서 모델의 성능 확인을 위해 몇 가지 데이터를 무작위로 선택하여 예측 및 실제 라벨 표시
    random_idx = random.sample(range(len(y_test)), 5)
    random_data = X_test[random_idx]
    random_answer = y_test[random_idx]
    print("------Prediction Data-----")
    predictions = model.predict(random_data)
    rounded_predictions = np.round(predictions, 2)
    print("Predictions:₩t", rounded_predictions)          # 예측
    print("Labels:₩t₩t", list(random_answer))             # 실제

    # 시각화
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plot_draw(y_train, train_predictions, 'Train Set with Best Parameters')
    plt.subplot(1, 2, 2)
    plot_draw(y_test, y_pred, 'Test Set with Best Parameters')
    plt.show()
    # Learing Curve 그리기
    plot_learning_curve(best_model, "Learning Curve", X, y, cv=5)
```

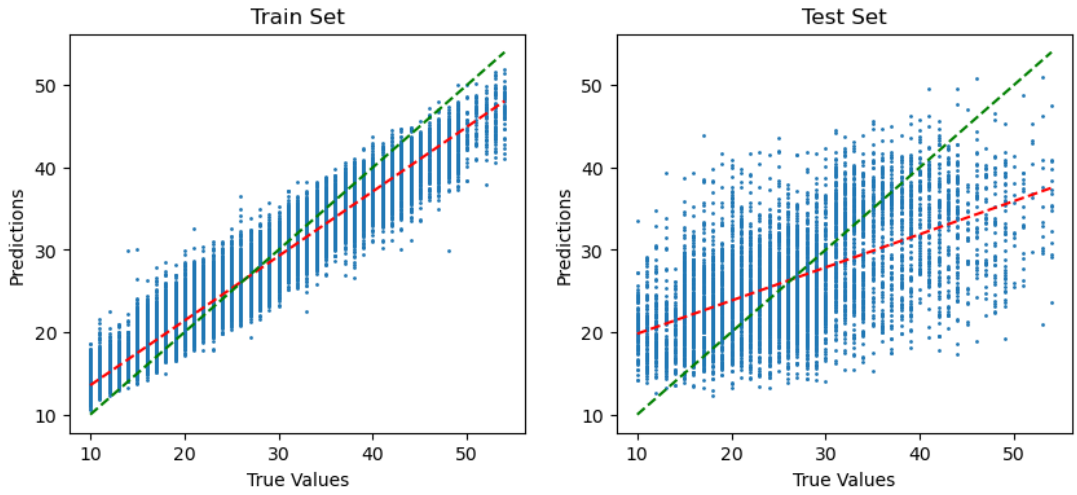
검증함수와 비슷하지만 최적 하이퍼 파라미터를 계산하는 부분이 추가되었음. 최적 하이퍼 파라미터를 조절하는 것이 꼭 큰 성능향상을 보이는 것은 아니지만, 비선형 알고리즘 중 Decision Tree 나 Random Forest 는 파라미터에 따라 과적합을 방지할 수 있기에 파라미터 조절은 필수라고 생각하였음. GridSerachCV 를 이용하였고 MSE 를 평가 지표로 사용하였으며 n_jobs 을 -1 로 설정해 CPU 의 모든 코어를 활용하여 학습을 더 빠르게 할 수 있도록 조정하였음.

데이터 학습 결과

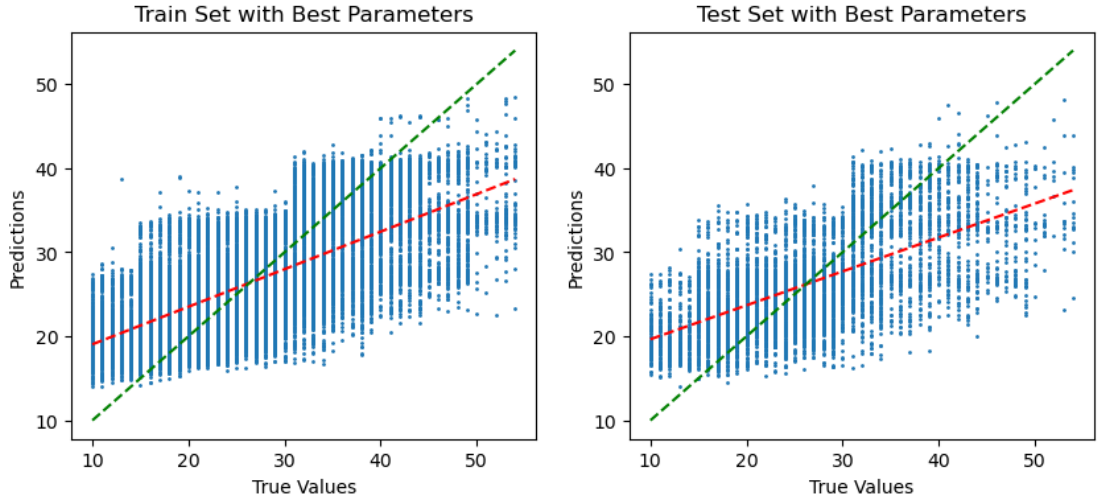
선형 알고리즘

Linear Regression

```
------Train Indicator-----
MSE: 8.493007082179465
MAE: 2.2550159593253967
R² : 0.9041080699361502
------Test Indicator-----
MSE: 60.80187003363068
MAE: 6.096204202380952
R² : 0.2998567270941O816
------Prediction Data-----
Predictions:      [17.76 31.4  14.94 32.33 36.78]
Labels:           [32, 25, 12, 21, 29]
```

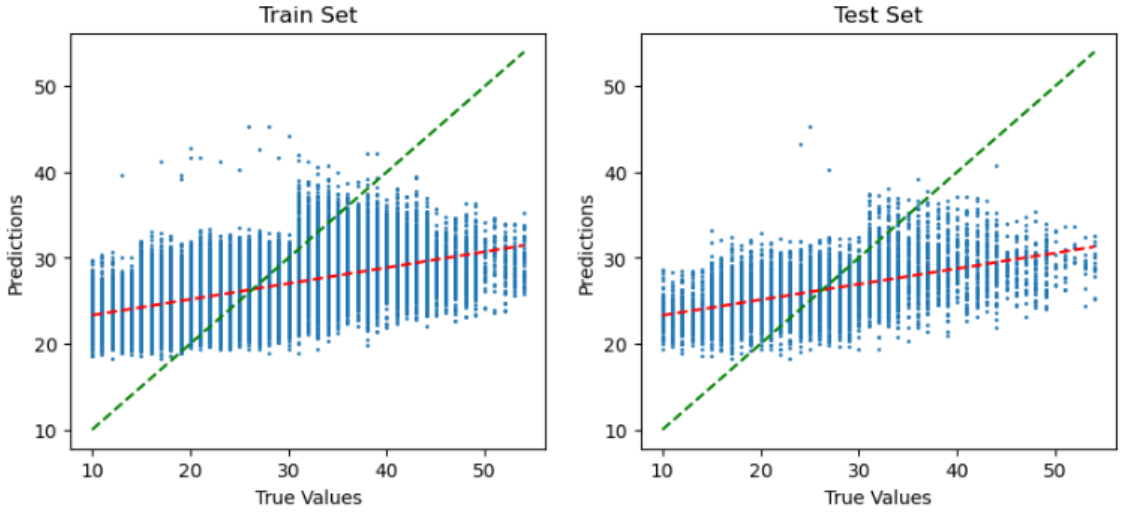



```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 4, 'n_estimators': 50}
------Train Accuracy with Best Parameters-----
MSE: 46.94191679047324
MAE: 5.384514632154104
R² : 0.46999325935096403
------Test Accuracy with Best Parameters-----
MSE: 52.93445692727022
MAE: 5.720354374943871
R² : 0.3904512492452059
------Prediction Data-----
Predictions:      [28.86 16.88 33.41 26.31 26.3 ]
Labels:           [34, 11, 36, 16, 12]
```

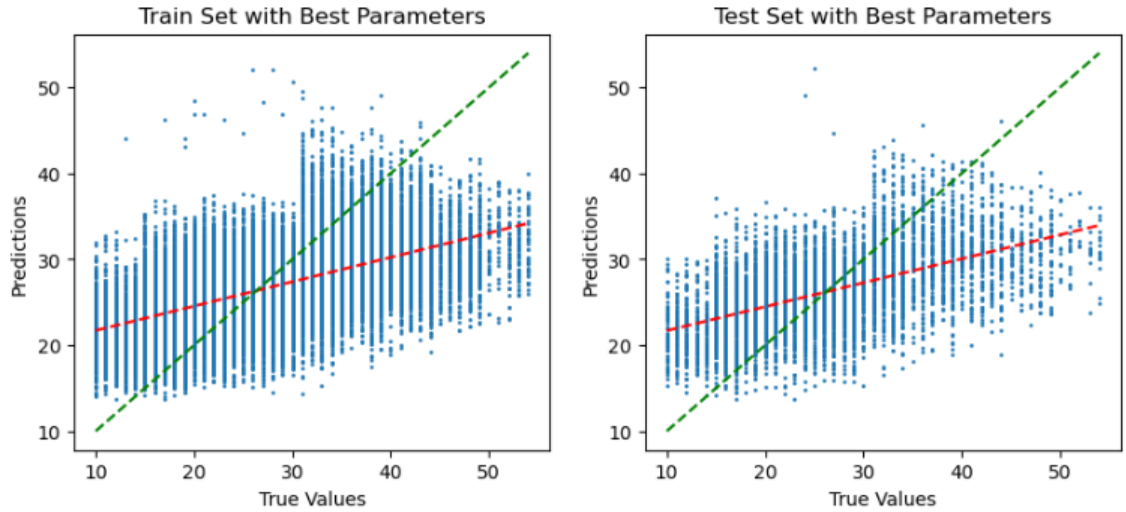


Lasso

```
------Train Indicator-----
MSE: 66.43834281994515
MAE: 6.516079152394144
R² : 0.24986511119058608
------Test Indicator-----
MSE: 65.49314379795939
MAE: 6.440956146719557
R² : 0.245835958232264
------Prediction Data-----
Predictions:      [27.72 25.07 26.39 27.16 26.34]
Labels:           [16, 36, 25, 36, 29]
```

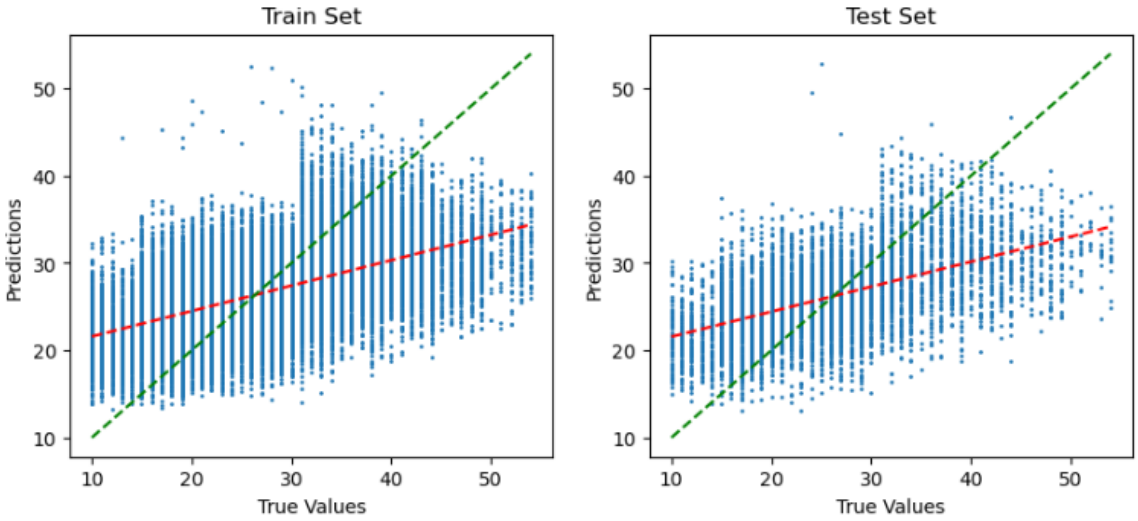



```
Best Parameters: {'alpha': 0.07}
------Train Accuracy with Best Parameters-----
MSE: 62.79752017756063
MAE: 6.314010922061375
R² : 0.29097251953491365
------Test Accuracy with Best Parameters-----
MSE: 61.98696706550483
MAE: 6.245600987086237
R² : 0.28621014493884744
------Prediction Data-----
Predictions:      [29.72 24.9  23.6  25.76 21.99]
Labels:           [27, 22, 24, 19, 21]
```

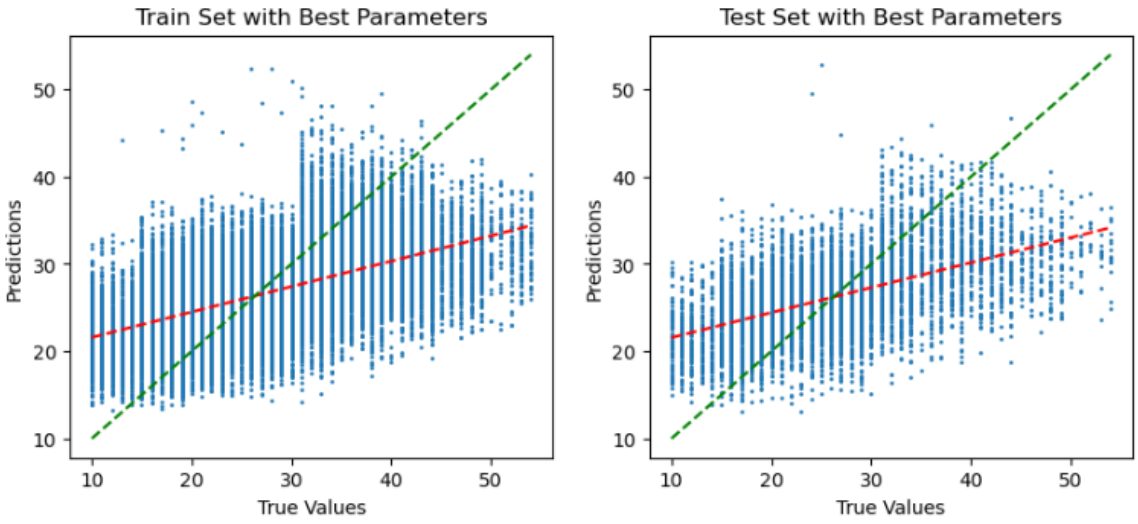


Ridge

```
------Train Indicator-----
MSE: 62.76938523487153
MAE: 6.311263624900678
R² : 0.29129018251701033
------Test Indicator-----
MSE: 61.96096252278008
MAE: 6.2430387645899295
R² : 0.2865095914138269
------Prediction Data-----
Predictions:      [24.07 22.97 38.03 20.74 23.73]
Labels:           [42, 30, 35, 18, 25]
```

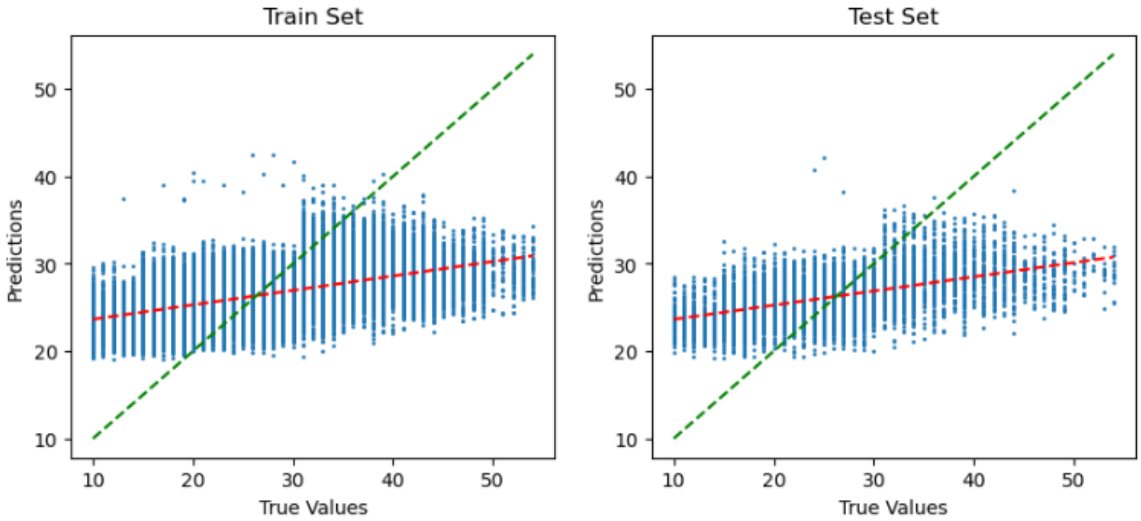



```
Best Parameters: {'alpha': 10}
------Train Accuracy with Best Parameters-----
MSE: 62.76938855175837
MAE: 6.311295928977548
R² : 0.2912901450670615
------Test Accuracy with Best Parameters-----
MSE: 61.96097822467515
MAE: 6.243076525657146
R² : 0.28650941060398993
------Prediction Data-----
Predictions:      [27.37 25.72 29.09 26.48 33.14]
Labels:           [15, 49, 24, 17, 47]
```

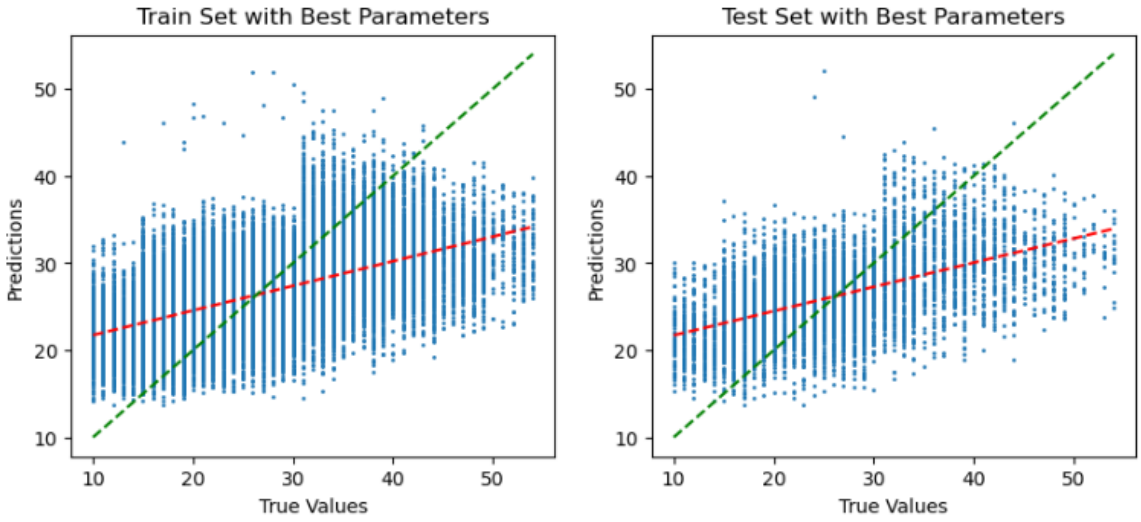


Elastic Net

```
------Train Indicator-----
MSE: 67.64186768639128
MAE: 6.596938281616123
R² : 0.23627648219185227
------Test Indicator-----
MSE: 66.66636412169943
MAE: 6.519271973224854
R² : 0.2323261382736227
------Prediction Data-----
Predictions:     [27.34 31.35 27.3  30.84 23.42]
Labels:          [33, 36, 20, 38, 21]
```



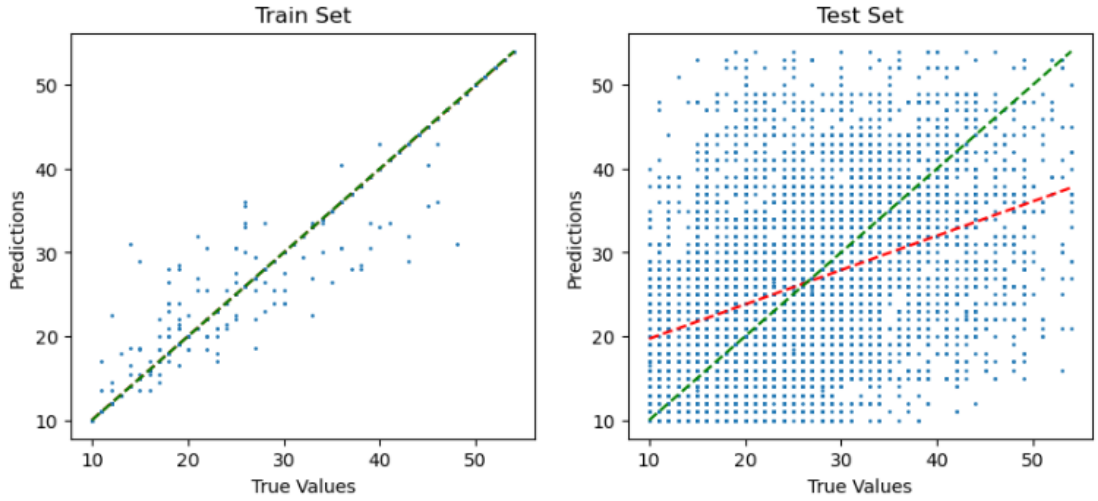

```
Best Parameters: {'alpha': 0.07, 'l1_ratio': 0.9}
------Train Accuracy with Best Parameters-----
MSE: 62.80074974675299
MAE: 6.314530282538782
R² : 0.29093605546275014
------Test Accuracy with Best Parameters-----
MSE: 61.99025957221827
MAE: 6.246195300574713
R² : 0.28617223119663626
------Prediction Data-----
Predictions:     [24.02 26.78 29.2  24.43 29.5 ]
Labels:          [18, 21, 37, 23, 32]
```



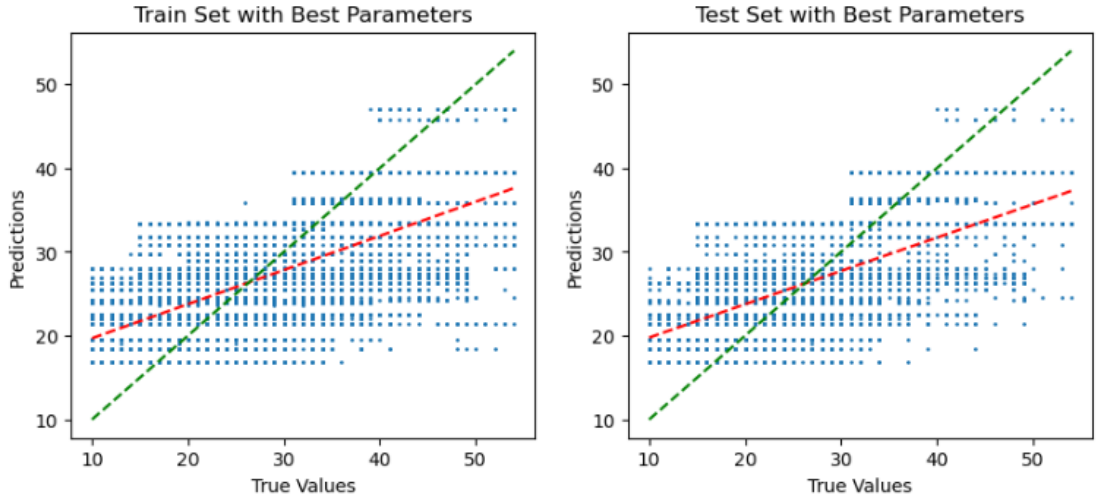
비선형 알고리즘

Decision Tree

```
------Train Indicator-----
MSE: 0.16719444444444445
MAE: 0.022444444444444447
R² : 0.9981122589656882
------Test Indicator-----
MSE: 108.125875
MAE: 8.021416666666667
R² : -0.2450867706279467
------Prediction Data-----
Predictions:     [18. 22. 31. 29. 21.]
Labels:          [23, 15, 23, 10, 30]
```

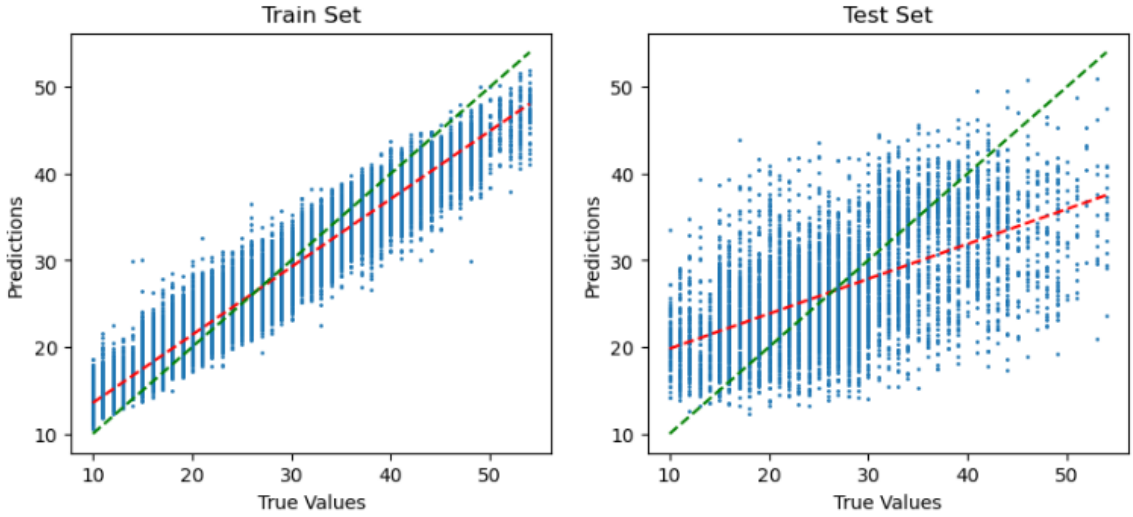



```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 5}
------Train Accuracy with Best Parameters-----
MSE: 52.50637462911445
MAE: 5.712932458993181
R² : 0.40716667781827864
------Test Accuracy with Best Parameters-----
MSE: 52.650661523251785
MAE: 5.70183655295173
R² : 0.39371919878187667
------Prediction Data-----
Predictions:     [33. 19. 18. 22. 30.]
Labels:          [33, 19, 28, 21, 28]
```

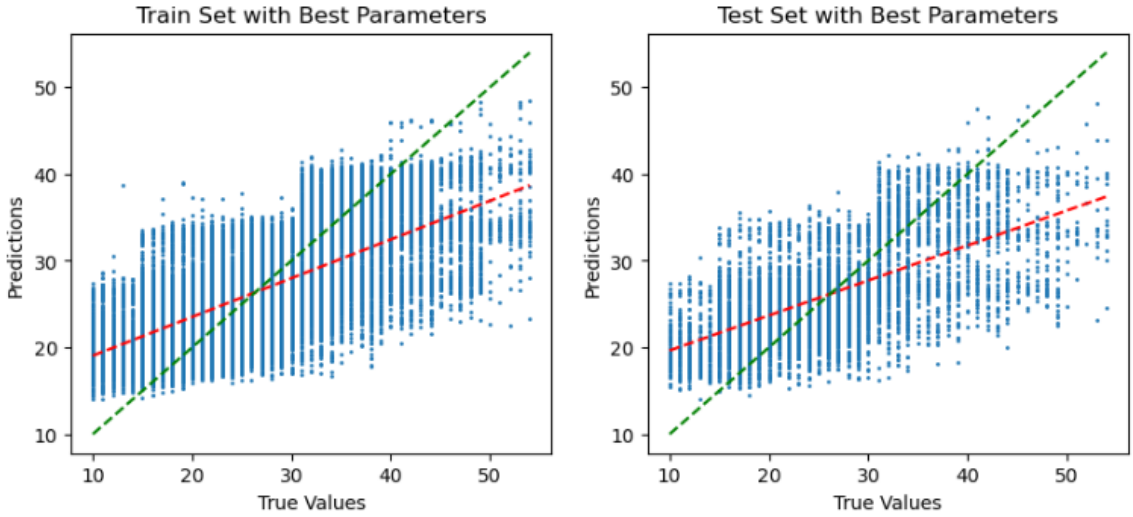


Random Forest

```
------Train Indicator-----
MSE: 8.493007082179465
MAE: 2.2550159593253967
R² : 0.9041080699361502
------Test Indicator-----
MSE: 60.80187003363068
MAE: 6.096204202380952
R² : 0.2998567270941O816
------Prediction Data-----
Predictions:      [17.76 31.4  14.94 32.33 36.78]
Labels:           [32, 25, 12, 21, 29]
```

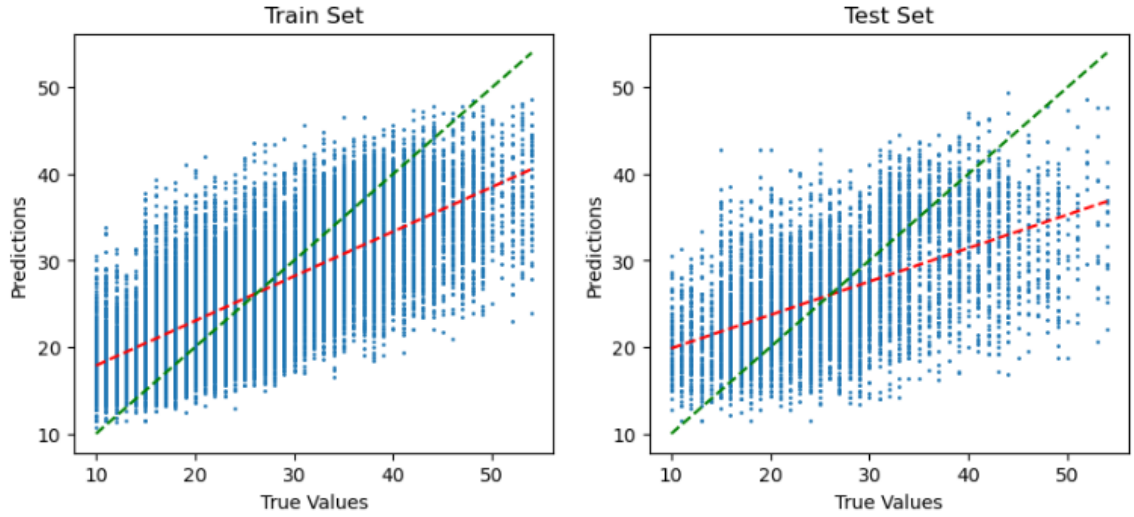



```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 4, 'n_estimators': 50}
------Train Accuracy with Best Parameters-----
MSE: 46.94191679047324
MAE: 5.384514632154104
R² : 0.46999325935096403
------Test Accuracy with Best Parameters-----
MSE: 52.93445692727022
MAE: 5.720354374943871
R² : 0.3904512492452059
------Prediction Data-----
Predictions:      [28.86 16.88 33.41 26.31 26.3 ]
Labels:           [34, 11, 36, 16, 12]
```

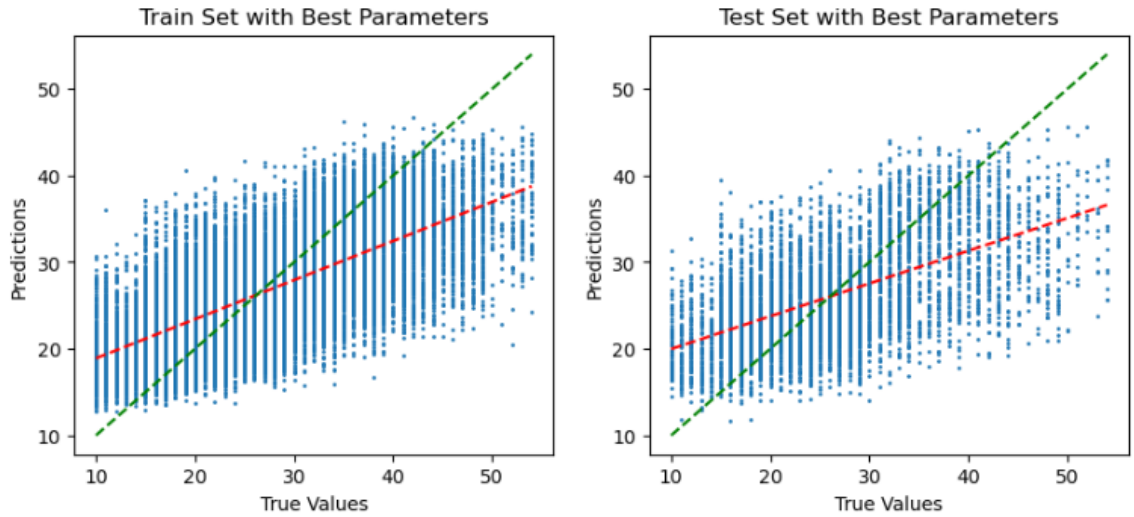


KNN

```
------Train Indicator-----
MSE: 42.63998333333333
MAE: 5.084533333333334
R² : 0.518565066511819
------Test Indicator-----
MSE: 63.37382000000001
MAE: 6.208100000000001
R² : 0.2702403112468982
------Prediction Data-----
Predictions:      [26.2 19.  30.8 27.4 30.2]
Labels:           [38, 22, 17, 14, 18]
```

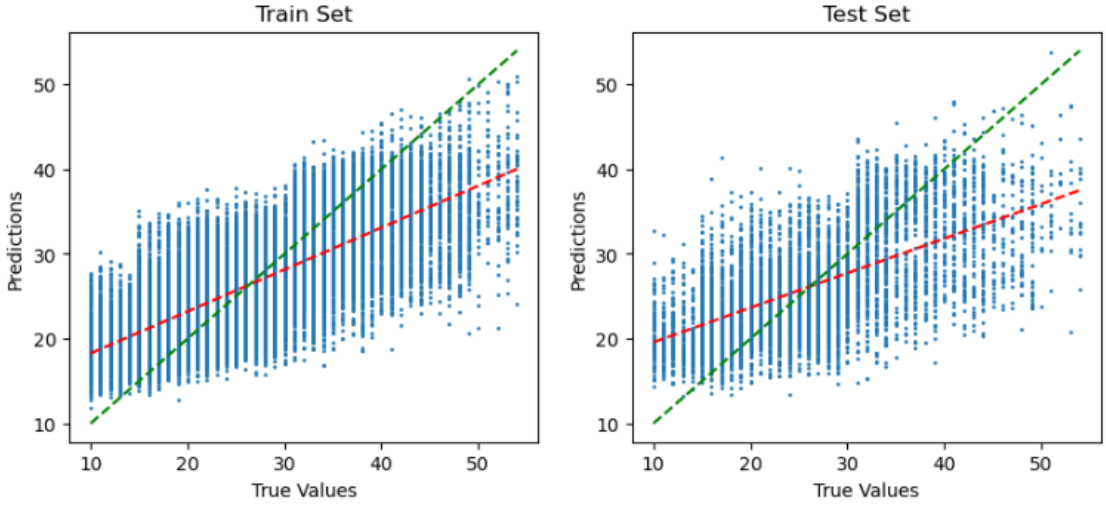



```
Best Parameters: {'n_neighbors': 9, 'p': 1, 'weights': 'uniform'}
------Train Accuracy with Best Parameters-----
MSE: 47.72232098765433
MAE: 5.41962037037037
R² : 0.4611819556544624
------Test Accuracy with Best Parameters-----
MSE: 58.80858024691358
MAE: 6.001148148148149
R² : 0.3228097782806929
------Prediction Data-----
Predictions:      [30.  23.8 25.4 15.6 35.4]
Labels:           [37, 17, 22, 29, 38]
```

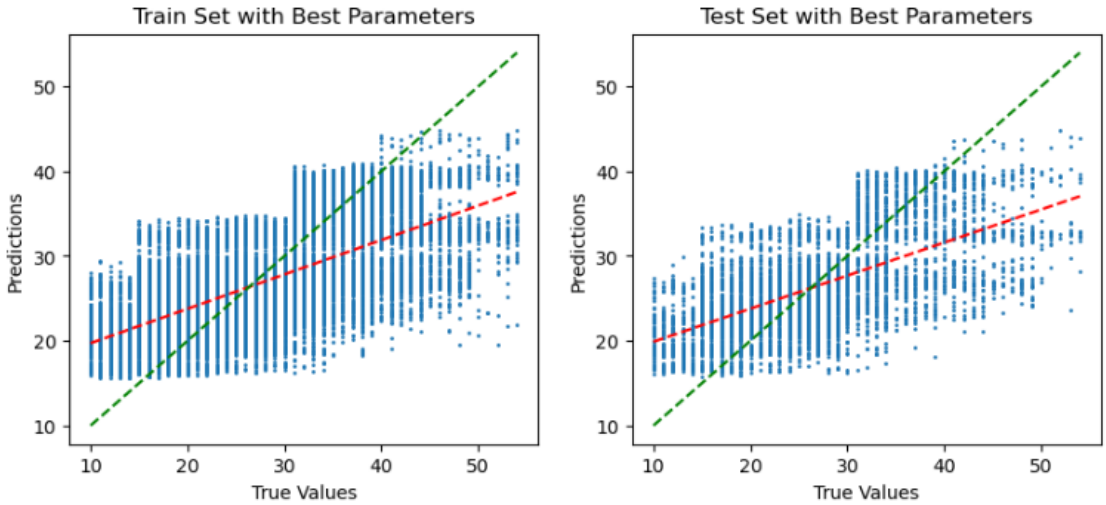


XGB

```
------Train Indicator-----
MSE: 41.30726804739779
MAE: 5.019972630500794
R² : 0.5336123447911584
------Test Indicator-----
MSE: 54.72886639718129
MAE: 5.791066642602285
R² : 0.369788336764028
------Prediction Data-----
Predictions:     [18.36 19.07 24.83 22.88 26.53]
Labels:          [11, 22, 29, 16, 18]
```

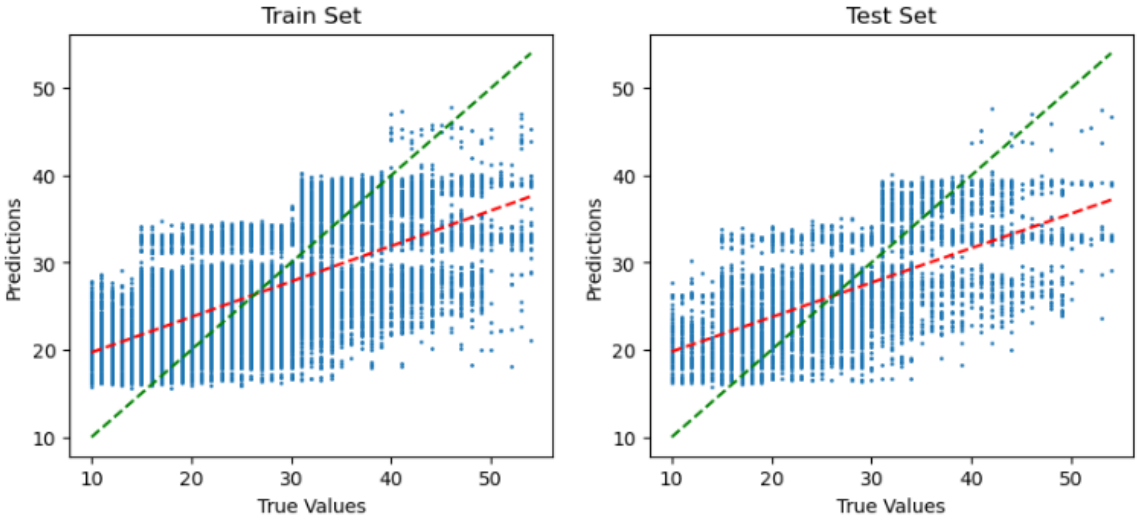



```
Best Parameters: {'colsample_bytree': 0.5, 'learning_rate': 0.07, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.8}
------Train Accuracy with Best Parameters-----
MSE: 50.96964057905694
MAE: 5.622112144351005
R² : 0.42451746919628786
------Test Accuracy with Best Parameters-----
MSE: 52.26372387437401
MAE: 5.6838667953809106
R² : 0.39817484778980927
------Prediction Data-----
Predictions:     [25.47 28.4  33.63 29.89 17.83]
Labels:          [38, 28, 27, 31, 16]
```

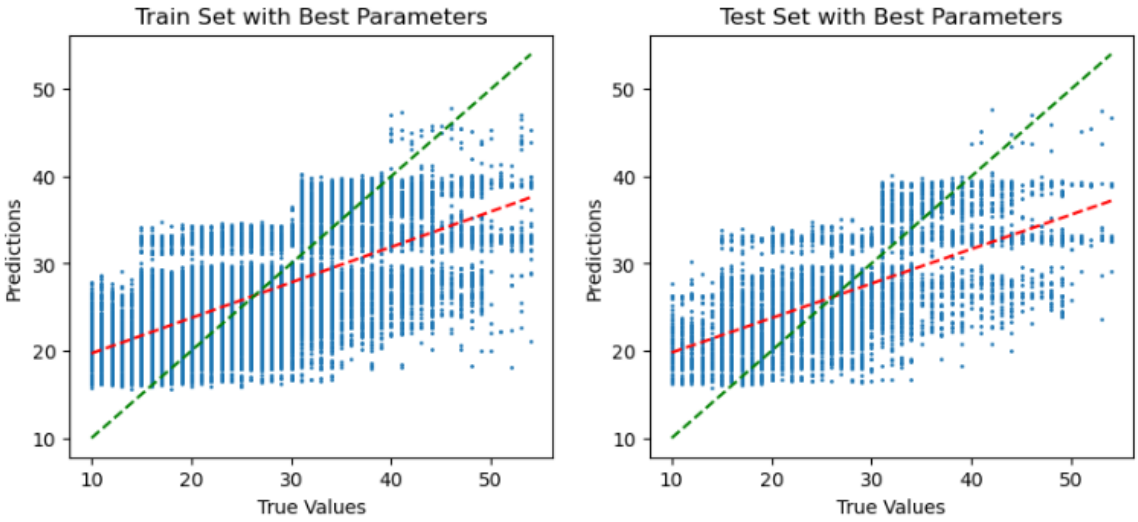


GBR

```
------Train Indicator-----
MSE: 51.72762521716273
MAE: 5.6654295767635805
R² : 0.41595929784385555
------Test Indicator-----
MSE: 52.31790046203511
MAE: 5.682165078468029
R² : 0.39755099570468533
------Prediction Data-----
Predictions:     [22.15 21.25 23.85 22.32 28.22]
Labels:          [24, 28, 20, 18, 31]
```

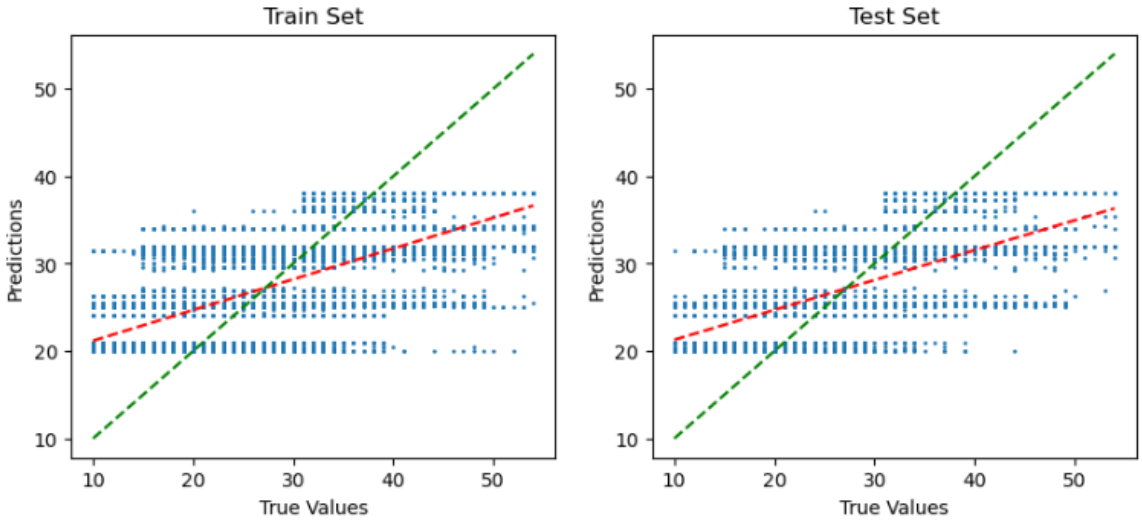



```
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 100}
------Train Accuracy with Best Parameters-----
MSE: 51.72762521716273
MAE: 5.6654295767635805
R² : 0.41595929784385555
------Test Accuracy with Best Parameters-----
MSE: 52.31790046203511
MAE: 5.682165078468029
R² : 0.39755099570468533
------Prediction Data-----
Predictions:     [22.06 17.12 28.1  33.7  18.39]
Labels:          [15, 24, 23, 33, 15]
```

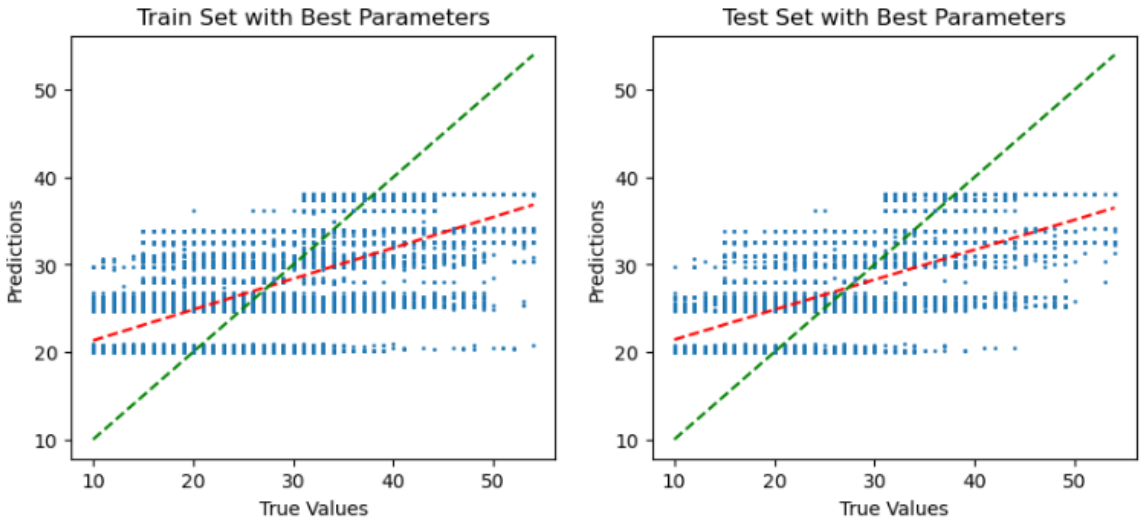


Adaboost

```
------Train Indicator-----
MSE: 56.32523448270016
MAE: 5.994696286651417
R² : 0.36404910609600105
------Test Indicator-----
MSE: 56.47870616940004
MAE: 5.988875253441892
R² : 0.3496386514911163
------Prediction Data-----
Predictions:     [31.92 21.01 20.19 25.55 33.98]
Labels:          [18, 17, 26, 20, 33]
```




```
Best Parameters: {'learning_rate': 0.01, 'n_estimators': 1000}
------Train Accuracy with Best Parameters-----
MSE: 56.34216345501988
MAE: 6.011006896788234
R² : 0.363857966277792
------Test Accuracy with Best Parameters-----
MSE: 56.528784585962846
MAE: 6.010210242691685
R² : 0.3490619905026495
------Prediction Data-----
Predictions:     [31.22 25.55 25.24 20.06 24.03]
Labels:          [48, 23, 18, 14, 20]
```



데이터 학습 결과 분석

데이터가 잘 학습되고 예측하였는지를 판단하는 지표로 MSE, MAE, R2 스코어를 활용하였음. 각각의 지표가 어떤 의미를 가지고 있는지 설명하자면,

MSE: 평균 제곱 오차, 실제 값과 예측 값 사이의 차이를 제곱하여 평균을 낸 값으로 오차의 크기를 과장하기 때문에 큰 오차가 있을 때 더 큰 패널티가 부여됨. RMSE는 MSE에 루트를 씌워 얻어낸 값임.

MAE: 평균 절대 오차, 실제 값과 예측 값 차이의 절댓값을 평균을 낸 값. 절대적인 크기만을 고려하기 때문에 MSE에 비해 큰 오차에 대해 덜 민감함.

R2 SCORE: 결정 계수, 모델이 데이터의 분산을 얼마나 잘 설명하는지를 나타내는 지표. 0에서 1 사이 값이며 1에 가까울수록 모델의 성능이 좋음을 나타냄.

MSE와 MAE를 바탕으로 모델이 과적합인지 과소적합인지를 예상할 수 있는데, 과적합의 경우에는 모델이 훈련데이터에 너무 적응되어 있어 새로운 데이터에서는 잘 일반화를 하지 못하는 상황을 말함. 훈련 세트의 MSE MAE가 낮지만, 테스트 세트의 MSE MAE가 월등하게 높으면 과적합을 의심할 수 있음. 역으로 훈련 세트의 MSE MAE가 월등하게 높으면 훈련 데이터에도 학습을 하지 못하는 과소적합을 의심할 수 있음.

또한 학습 곡선(Learning Curve)를 그림으로써 모델의 과적합, 과소적합의 징후를 판단하고 학습을 통해 성능이 향상되는지를 판단하려고 함.

결과 분석 전 먼저 모델들의 R2 스코어에 대해 설명하고자 함. 전반적으로 모델들의 R2 스코어가 낮으면 0.28 높으면 0.4 정도의 분포를 보이고 있는데, 이는 학습이 잘 안된 것이 아니라 학습이 잘 된 상황이라고 볼 수 있음. 분명 1과 가까워야 하는 R2 스코어가 0.5도 넘지 않는 상황에서 그렇게 판단한 이유가 무엇이냐면, **배달을 로봇이 하는 것이 아니라 사람이 하고 있기 때문임**. 데이터에서는 사람의 신체조건이나 성별을 명시하지 않고 있으며, 목적지까지 갈 때 신호에 걸리거나, 중간에 길을 잘못 들었을 가능성도 존재함. 그렇기에 사람이 하는 행동을 예측할 때에는 R2 SCORE가 높게 나오지 않는 경향이 있음.

(https://statisticsbyjim.com/regression/how-high-r-squared/)

(https://towardsdatascience.com/an-ode-to-r-squared-804d8d0ed22c)

R2 SCORE는 높지 않지만 MAE는 6.XX 대로 측정되고 있음. 이는 평균적인 절대 오차가 6분 정도임을 의미하는데 사용중인 데이터가 9분~54분대로 분포된 타겟을 예측한다는 것을 생각해본다면 오차가 큰 편이 아님을 확인할 수 있음.

선형 알고리즘

Linear Regression

초반 Train, Test Indicator를 살펴보자면 mae, mse 둘의 값이 62.7, 6.31/61.9, 6.24로 과적합이나 과소적합을 일으키는 상황은 아님을 알 수 있음. Prediction data를 살펴보아도 큰 오차 없이 Predict을 하고 있음을 확인할 수 있음. Linear Regression은 파라미터로 Fit_intercept와 Copy_X를 사용함. Fit_intercept는 회귀 선이 y축을 어디서 만나는지 고려할지를 선택하는 파라미터로 일반적으로 True를 사용하며 Copy_X는 입력데이터를 복사하여 사용할지 고려하는 파라미터임. 둘 다 성능향상에는 별 도움이 없는 파라미터들이기에 동일한 값을 얻었음. 학습곡선을 그려보았을 때 다음과 같이 성능이 향상됨을 확인할 수 있었음. 초반에는 Train Score가 Cross-val Score(Test Score) 보다 많이 낮은 과적합이 있었지만 train set의 크기가 커짐에 따라 이상적인 학습곡선으로 바뀜을 확인할 수 있음. 현재 총 데이터는 30000개며 train set을 0.8로 설정하였기에 24000개를 사용중인 상황임. 학습곡선을 보았을 때 이상적인 train set 수치를 설정했음을 확인할 수 있었음.

Lasso

Linear Regression보다 초반 mae, mse 수치는 높게 측정되어 상대적으로 좋지 않은 모델 학습률을 보였음. Lasso 모델은 alpha라는 계수를 사용하여 정규화의 강도를 조절함. Alpha가 크면 과소적합이 일어날 수 있으며 작으면 과적합이 일어날 수 있음. Alpha를 적당히 조절함으로써 중요하지 않은 특성의 계수를 0으로 만들어 해석하기 좋은 모델을 만들 수 있음. 최적 파라미터를 계산한 결과 0.07로 계산되었으며 기존 디폴트 값인 1보다 감소했음. 파라미터 튜닝 후 test set에 대한 predict mse, mae를 확인하면 mae가 6.24로 LinearModel보다 성능이 향상됨을 확인하였음.

Ridge

Ridge 모델 또한 Lasso 모델과 같이 선형 모델의 확장된 버전으로, 똑같이 alpha를 사용하여 파라미터 튜닝을 진행함. 이때 Ridge 모델은 Lasso 모델보다 파라미터 값의 범위가 더 큰데, 이는 두 모델이 회귀 계수에 적용하는 정규화 방식이 차이가 있기 때문임. Ridge 모델은 계수의 크기에 더 큰 가중치를, Lasso는 계수를 0으로 만드는 것에 더 집중을 한다는 차이가 있음. 최적 파라미터는 alpha를 10으로 계산하였으며, 최적 파라미터로 재학습을 시켰을 때 mae는 6.24로 Lasso와 비슷한 수치를 보였음

Elastic Net

Elastic Net은 Ridge 모델과 Lasso 모델의 특성을 결합한 모델임. 사용하는 파라미터 변수는 alpha와 l1_ratio로 alpha는 위에서와 같이 정규화의 전체 강도를 조절하는 변수이고, l1_ratio는 Elastic 모델이 Lasso와 Ridge중 어디로 중점을 줄지 선택하는 변수임. L1_ratio의 값이 0에 가까워지면 L2 정규화의 영향이 커지며 릿지 회귀에 가까워지게 됨. 하이퍼 파라미터 학습 결과 L1_ratio가 0.9로 Lasso 모델에 거의 근접하게 튜닝되었음을 확인할 수 있었음. 사용한 선형모델들은 거의 다 비슷한 결과값을 도출하였음.

비선형 알고리즘

Decision Tree

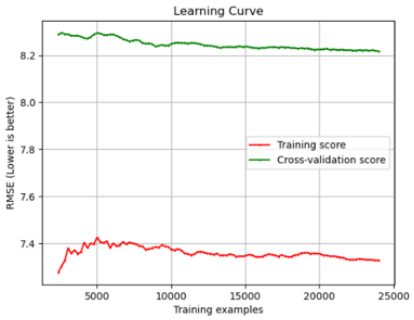
Decision Tree의 경우 파라미터 제한을 하지 않자 분류 모델을 학습할 때처럼, 심한 과적합을 보였으며 Test set에 대한 R2 score가 마이너스가 나왔음. Max_depth, min_samples_leaf, min_samples_split 변수를 활용해 파라미터 튜닝을 진행하였고 튜닝결과 mae가 5.7, R2 Score가 0.39로 꽤나 좋은 학습을 하고 있음을 알 수 있었음.

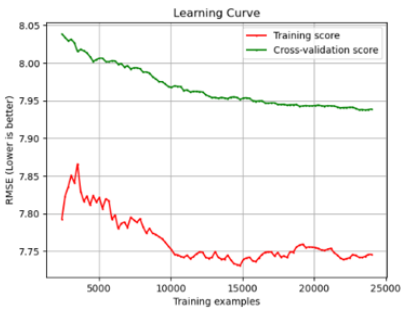
Random Forest

Decision Tree에서 발전한 모델로 역시 하이퍼파라미터 튜닝 전에는 Train set의 Mae가 2.25, Test set의 Mae가 6.09로 심한 과적합을 보였음. 튜닝결과 mae는 5.7, R2 Score는 0.39로 측정되었음. 하지만 기존의 다른 모델들과 다르게 학습곡선을 그려본 결과 Training Score와 Cross-val Score의 차이가 많이 남을 확인할 수 있었음. 하지만 Training examples수를 늘려감에 따라 둘의 차이가 감소함을 확인할 수 있었고 더 많은 데이터를 사용해서 학습했다면 좋은 결과를 낼 수 있었을 것이라고 생각함.

KNN

KNN은 분류모델만 있는줄 알았으나 회귀모델로도 사용할 수 있는 KNeighborsRegressor의 존재를 알게 되어 학습 해보았음. 파라미터 튜닝 전에는 과적합의 의심되는 수치를 확인할 수 있었고 파라미터 튜닝을 통해 이웃의 개수를 9로 정하여 이를 일부 해결할 수 있었음. 하지만 학습곡선을 그려본 결과 학습을 통해 모델의 학습성능이 과적합 되었으며 학습을 통해 개선되지 않음을 확인할 수 있었음. 이웃의 수가 너무 적으면 과적합을 일으키기에 파라미터의 범주를 0~9에서 10에서 50으로 수정한 후 2차 학습을 진행하였고, 새로운 이웃의 수인 40으로도 테스트를 해본 결과 과적합을 개선할 수는 없었음. KNN은 새로운 데이터에 가장 가까운 일부의 훈련 데이터를 찾고, 이들의 평균값으로 예측을 수행하는데 이러한 요소가 데이터에 맞지 않는 학습 모델이었다고 생각함.



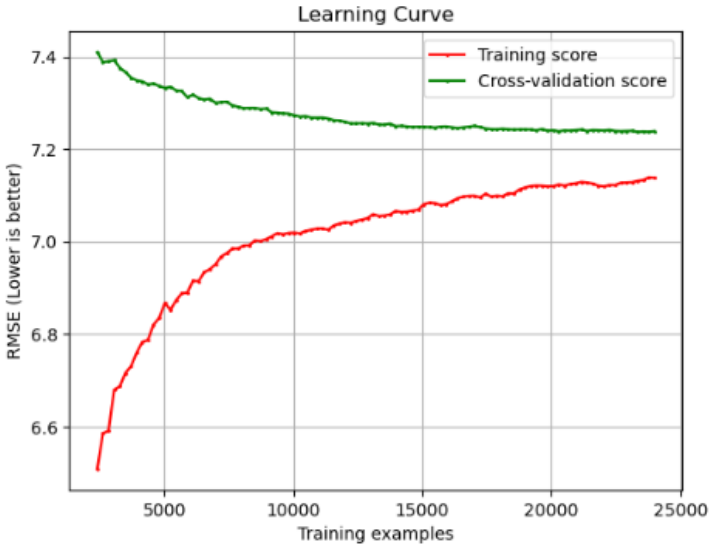


XGB GBR Adaboost

이 알고리즘들은 앙상블 학습을 구현하는 비선형 알고리즘으로, Decision Tree를 기본으로 학습함

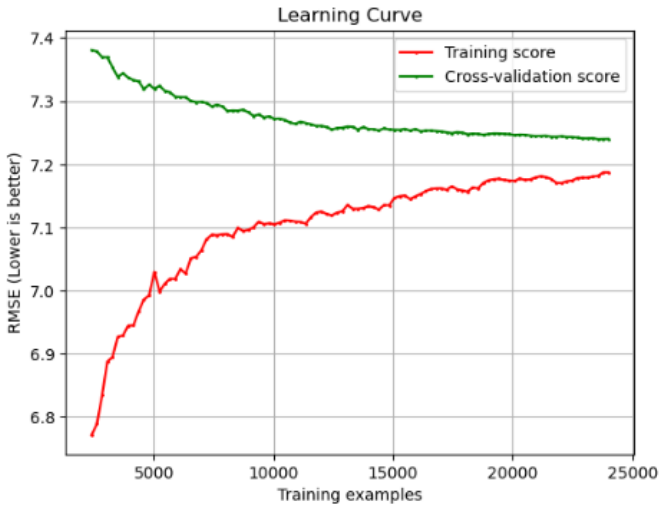
XGB

파라미터 학습 전에는 Train, Test set의 mae 차이가 0.79차이로 과적합을 의심할 수 있었음. 파라미터 학습 후 mae차이가 0.01차이로 줄어들었고 mse도 차이가 많이 줄어 문제가 해결됐음을 확인하였음. Rmse 수치도 사용하던 모델 중에서는 제일 낮게 측정되었음.



GBR

하이퍼 파라미터 튜닝 전에도 준수한 성적을 보였으며 과적합이 의심되지는 않음. 하이퍼 파라미터 튜닝을 진행하였으나 두 변수 모두 디폴트 값인 100, 0.1이 제일 최적의 하이퍼 파라미터로 계산되어 성능의 향상은 확인할 수 없었음.



Adaboost

Adaboost 알고리즘도 비슷하게 진행하였으며 하이퍼 파라미터 계산 결과 learning_rate는 0.01, n_estimators는 1000으로 측정되었음. n_estimators의 기본 디폴트 값은 50으로 수가 증가할수록 과적합의 위험이 있으나 다행히 과적합으로 의심되는 상황은 확인할 수 없었음.

결론적으로 선형회귀 모델들보다 비선형 모델들의 성능이 현 데이터에서는 더 좋았으며, 제일 좋은 모델은 XGB로 MAE 5.68로 측정되었음.

고찰 및 결론

처음에는 데이터를 표준화와 2차다항평가를 모두 진행하여 41500 * 77 사이즈의 데이터로 학습을 진행하였으나, 학습시간이 무한정으로 걸리기에 어쩔 수 없이 2차 다항평가를 제외하여 학습을 진행하였음. 비선형 알고리즘에 굳이 2차 다항평가 전처리를 한 데이터를 넣는 것이 좋을지 안좋을지는 모르겠으나 직접 학습을 돌려보고 판단할 수 있었다면 더 좋았을 것 같음

처음에 학습을 돌렸을 때 R2 Score가 0.3 수준으로 나와 굉장히 충격을 받았음. 이에 전처리를 잘못했나 하고 전처리를 하지 않은 Raw data의 학습을 진행한 결과 0.19의 점수를 확인할 수 있었음. 이에 전처리의 문제가 아니라 학습 데이터의 특성이 아닐까? 하는 생각을 하였고 검색을 통해 이 궁금증을 해결할 수 있었음. 비슷한 배달 데이터로 학습한 사람들의 R2 SCORE는 대부분 0.3을 넘지 못함을 확인할 수 있었고, (https://buly.kr/FsFHhs7)(https://buly.kr/31Py6LR) R2 SCORE에 대한 자료를 찾아 사람의 행동이 들어가는 데이터는 0.5를 넘지 못한다는 것을 확인한 후 안심하며 전처리와 학습이 굉장히 잘 되고 있음을 확인하였음.