BITAMIN 14주차 정규 Session

다양한 회귀 알고리즘

TEAM

비타민 6기 6조

TEAM MEMBERS

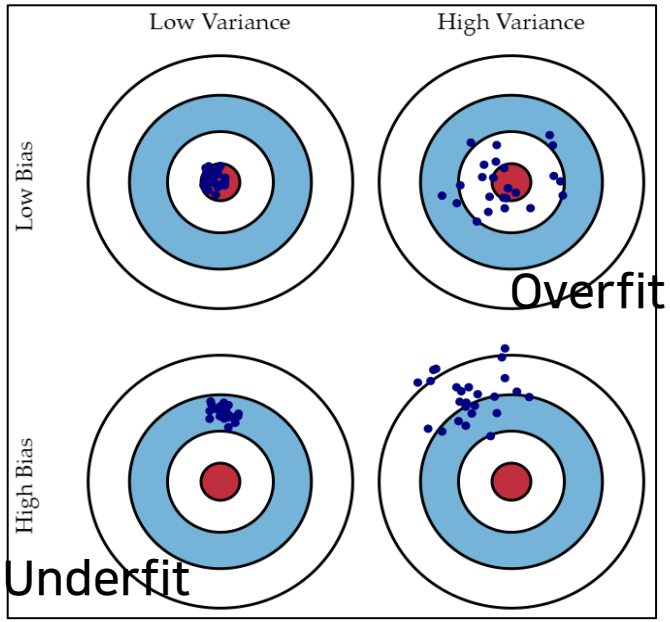
김중훈 김현정 백주은 서현재

Start

CONTENTS

CHAPTER.0

편향과 분산



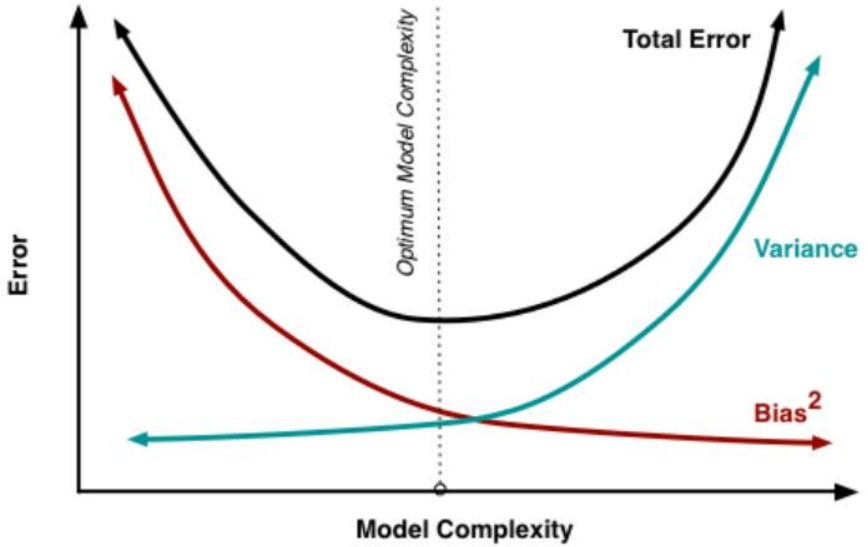
✓ 편향(Bias)

- 예측값과 실측값의 차이
- 지나치게 단순한 모델로 인한 error
- Under-fitting

✓ 분산(Variance)

- 모델이 예측한 값의 변동성
- 지나치게 복잡한 모델로 인한 error
- Over-fitting

✓ 편향-분산 Trade-off



- 모델의 복잡도 ↑ → 분산 ↑, 편향 ↓

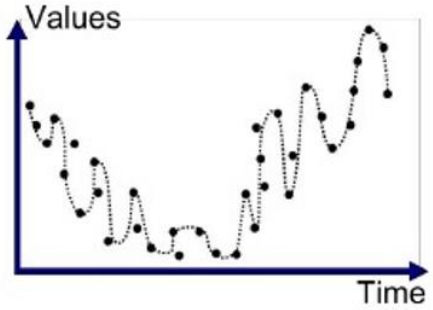
1. 규제 선형 모델 – 릿지, 라쏘, 엘라스틱

규제 선형 모델

✓ 비용함수의 목표

$$\boldsymbol{Min} \quad RSS(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (w_0 + w_1 * x_i)\right)^2$$

회귀 계수 값 ↑



Overfitted

최적 모델을 위한
Cost 함수 구성요소

학습데이터 잔차
오류 최소화

+

회귀계수 크기 제어

규제 선형 모델

규제(Regulation)

회귀함수 크기제어

$$Min \quad RSS(W) + \text{Regulation}$$

✓ L1 규제 $\quad alpha * \|W\|_1$

→ 라쏘 회귀

✓ L2 규제 $\quad alpha * \|W\|_2^2$

→ 릿지 회귀

✓ alpha

if. alpha = 0

비용함수
$=Min(RSS(W) + 0)$

회귀 계수 W 값이 커져도
어느정도 상쇄 가능

if. alpha = ∞

비용함수
$=Min(RSS(W) + \|W\|_1 \; or \|W\|_2^2)$

회귀 계수 W 값을 작게 하여
과적합 개선

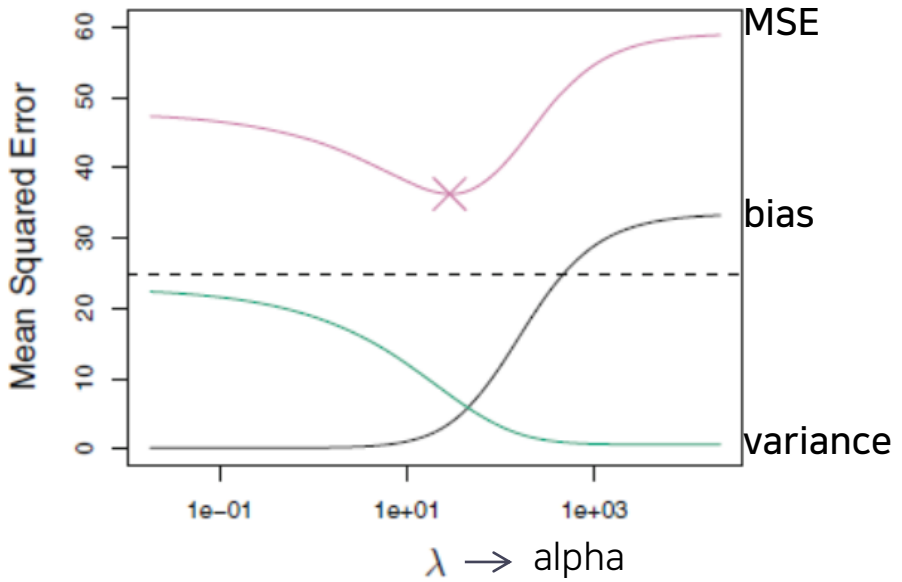
규제 선형 모델

릿지 회귀(Ridge)

상대적으로 큰 회귀 계수 값의 예측 영향도를 감소시키기 위해 회귀 계수값을 더 작게 만드는 규제 모델(L2규제)

$$\boldsymbol{Min} \qquad RSS(W) + \boldsymbol{alpha} * \|\boldsymbol{W}\|_2^2$$

- ✓ 영향이 적은 변수들에 대해 가중치를 0에 가까게 줌 → w↓
- ✓ 다중공선성(독립 변수들간의 강한 상관관계) 방지
- ✓ 계수 축소에 의해 모델의 복잡도 ↓ → 오버피팅 ↓
- ✓ 변수의 scaling 중요



- ✓ alpha값이 증가함에 따라 계수가 수축하여 변동이 크지 않은 모델이 됨

출처 : https://godongyoung.github.io/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D/2018/02/07/ISL-Linear-Model-Selection-and-Regularization_ch6.html

규제 선형 모델

릿지 회귀(Ridge)

✓ 보스턴 주택 가격 데이터를 이용한 실습(주택 가격 예측)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston

#보스턴 데이터 로드
boston = load_boston()

#보스턴 데이터 세트 DataFrame 변환
bostonDF = pd.DataFrame(boston.data, columns = boston.feature_names)

# boston 데이터 세트의 target 배열은 주택 가격. 이를 PRICE 칼럼으로 DataFrame에 추가.
bostonDF['PRICE'] = boston.target
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Target

→ 가격

```
from sklearn.model_selection import train_test_split

y_target = bostonDF['PRICE']
X_data = bostonDF.drop(['PRICE'], axis=1, inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_data, y_target, test_size=0.3,
                                                    random_state=156)
```

규제 선형 모델

릿지 회귀(Ridge)

✓ 보스턴 주택 가격 데이터를 이용한 실습(주택 가격 예측)

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score

# alpha=10으로 설정해 릿지 회귀 수행.
ridge = Ridge(alpha=10)
neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring="neg_mean_squared_error", cv=5)
rmse_scores = np.sqrt(-1*neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print('5 folds 의 개별 Negative MSE scores : ', np.round(neg_mse_scores, 3))
print('5 folds 의 개별 RMSE scores : ', np.round(rmse_scores, 3))
print('5 folds 의 평균 RMSE : {0:.3f}'.format(avg_rmse))
```

✓ 사이킷런에선 Ridge 클래스를 통해 릿지 회귀 구현

✓ 주요 생성파라미터 → alpha : L2 규제 계수

alpha는 10으로 가정
Ridge클래스를 이용한 예측
cross_val_socre( )로 평가

```
5 folds 의 개별 Negative MSE scores :  [-11.422 -24.294 -28.144 -74.599 -28.517]
5 folds 의 개별 RMSE scores :  [3.38  4.929 5.305 8.637 5.34 ]
5 folds 의 평균 RMSE : 5.518
```

지난 주 실습한 규제 없는 LinearRegression의 RMSE 평균 : 5.836

규제 선형 모델

릿지 회귀(Ridge)

✓ alpha값에 따른 변화

$$Min \quad RSS(W) + alpha * ||W||_2^2$$

RSS(W) 최소화 — *alpha* — 회귀 계수 W 감소

```
# 릿지에 사용될 alpha 파라미터의 값을 정의
alphas = [0, 0.1, 1, 10, 100]

# alphas list 값을 반복하면서 alpha에 따른 평균 rmse를 구함
for alpha in alphas:
    ridge = Ridge(alpha=alpha)

    # cross_val_score를 이용해 5 폴드의 평균 RMSE를 계산
    neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring="neg_mean_squared_error", cv=5)
    avg_rmse = np.mean(np.sqrt(-1*neg_mse_scores))
    print('alpha {0} 일 때 5 folds 의 평균 RMSE : {1:.3f}'.format(alpha, avg_rmse))
```

```
alpha 0 일 때 5 folds 의 평균 RMSE : 5.829
alpha 0.1 일 때 5 folds 의 평균 RMSE : 5.788
alpha 1 일 때 5 folds 의 평균 RMSE : 5.653
alpha 10 일 때 5 folds 의 평균 RMSE : 5.518
alpha 100 일 때 5 folds 의 평균 RMSE : 5.330
```

alpha 값이 증가함에 따라 RMSE 값 감소

규제 선형 모델

릿지 회귀(Ridge)

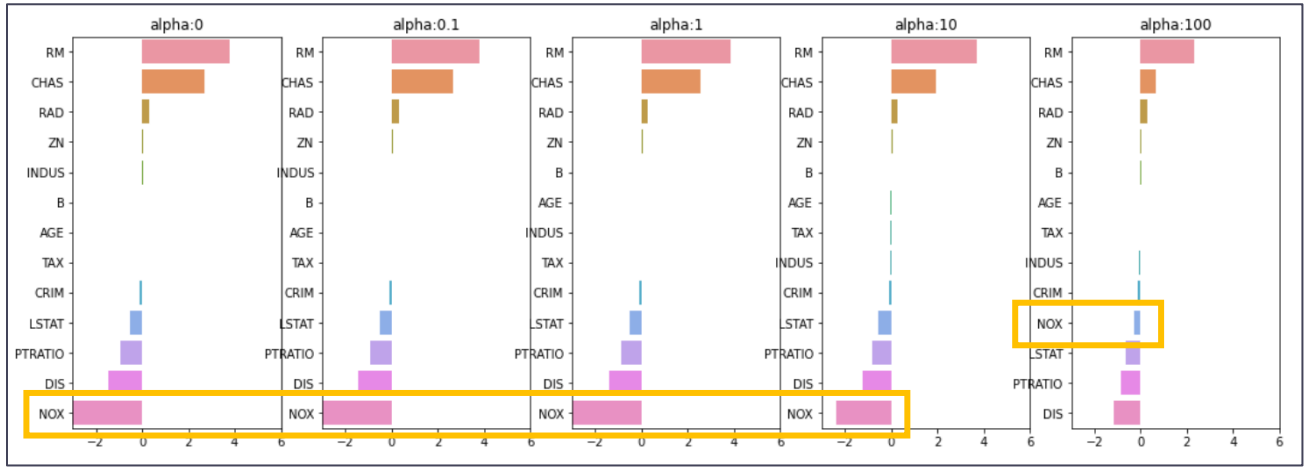
✓ alpha값에 따른 변화-그래프

```
#각 alpha에 따른 회귀 계수 값을 시각화하기 위해 5개의 열로 된 matplotlib 축 생성
fig, axs = plt.subplots(figsize=(18,6), nrows=1, ncols=5)
#각 alpha에 따른 회귀 계수 값을 데이터로 저장하기 위한 DataFrame 생성
coeff_df = pd.DataFrame()

# alphas 리스트 값을 차례로 입력해 회귀 계수 값 시각화 및 데이터 저장. pos는 axis의 위치 지정
for pos, alpha in enumerate(alphas):
    ridge = Ridge(alpha = alpha)
    ridge.fit(X_data, y_target)
    # alpha에 따른 피처별로 회귀 계수를 Series로 변환하고 이를 DataFrame의 칼럼으로 추가.
    coeff = pd.Series(data=ridge.coef_, index=X_data.columns)
    colname='alpha:'+str(alpha)
    coeff_df[colname] = coeff
    # 막대 그래프로 각 alpha 값에서의 회귀 계수를 시각화. 회귀 계수값이 높은 순으로 표현
    coeff = coeff.sort_values(ascending=False)
    axs[pos].set_title(colname)
    axs[pos].set_xlim(-3, 6)
    sns.barplot(x=coeff.values, y=coeff.index, ax=axs[pos])

# for 문 바깥에서 matplotlib의 show 호출 및 alpha에 따른 피처별 회귀계수를 DataFrame으로 표시
plt.show()
```

규제 선형 모델

릿지 회귀(Ridge)

✓ alpha값에 따른 변화-그래프



- alpha 값이 증가함에 따라 회귀 계수 값이 작아짐을 확인
- 특히 NOX 피처의 계수가 크게 작아짐

→ 하지만 계수가 0이 되지는 않음

✓ alpha값에 따른 회귀 계수 값 변화

```
ridge_alphas = [0, 0.1, 1, 10, 100]
sort_column = 'alpha:'+str(ridge_alphas[0])
coeff_df.sort_values(by=sort_column, ascending=False)
```

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100
RM	3.809865	3.818233	3.854000	3.702272	2.334536
CHAS	2.686734	2.670019	2.552393	1.952021	0.638335
RAD	0.306049	0.303515	0.290142	0.279596	0.315358
ZN	0.046420	0.046572	0.047443	0.049579	0.054496
INDUS	0.020559	0.015999	-0.008805	-0.042962	-0.052826
B	0.009312	0.009368	0.009673	0.010037	0.009393
AGE	0.000692	-0.000269	-0.005415	-0.010707	0.001212
TAX	-0.012335	-0.012421	-0.012912	-0.013993	-0.015856
CRIM	-0.108011	-0.107474	-0.104595	-0.101435	-0.102202
LSTAT	-0.524758	-0.525966	-0.533343	-0.559366	-0.660764
PTRATIO	-0.952747	-0.940759	-0.876074	-0.797945	-0.829218
DIS	-1.475567	-1.459626	-1.372654	-1.248808	-1.153390
NOX	-17.766611	-16.684645	-10.777015	-2.371619	-0.262847

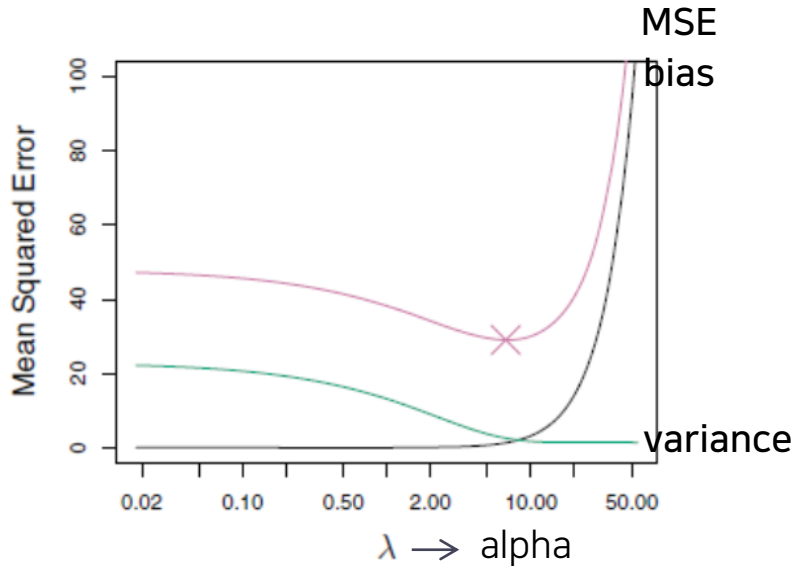
규제 선형 모델

라쏘 회귀(Lasso)

예측 영향력이 작은 피처의 회귀 계수를 0으로 만들어 회귀 예측 시 피처가 선택되지 않게 하는 규제 모델(L1 규제)

$$\boldsymbol{Min} \quad RSS(W) + \boldsymbol{alpha} * \|\boldsymbol{W}\|_{\boldsymbol{1}}$$

- ✓ 영향이 적은(불필요한) 변수의 회귀 계수를 급격하게 감소시켜 0으로 만듬
- ✓ 변수를 자동으로 채택함으로써 일반적으로 많은 변수를 다룰 때 활용
- ✓ 변수 제거에 따라 모델의 가장 중요한 특성을 앎 → 해석력 ↑



- ✓ 릿지와 동일하게 alpha값이 증가함에 따라 bias ↑, variance ↓

규제 선형 모델

라쏘 회귀(Lasso)

✓ alpha 값 변화에 따른 결과 출력 함수 만들기

```
from sklearn.linear_model import Lasso, ElasticNet

# alpha값에 따른 회귀 모델의 폴드 평균 RMSE를 출력하고 회귀 계수값들을 DataFrame으로 반환
def get_linear_reg_eval(model_name, params=None, X_data_n=None, y_target_n=None,
                        verbose=True, return_coeff=True):
    coeff_df = pd.DataFrame()
    if verbose : print('####### ', model_name , '#######')
    for param in params:
        if model_name =='Ridge': model = Ridge(alpha=param)
        elif model_name =='Lasso': model = Lasso(alpha=param)
        elif model_name =='ElasticNet': model = ElasticNet(alpha=param, l1_ratio=0.7)
        neg_mse_scores = cross_val_score(model, X_data_n,
                                         y_target_n, scoring="neg_mean_squared_error", cv = 5)
        avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
        print('alpha {0}일 때 5 폴드 세트의 평균 RMSE: {1:.3f} '.format(param, avg_rmse))
        # cross_val_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

        model.fit(X_data_n , y_target_n)
        if return_coeff:
            # alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가.
            coeff = pd.Series(data=model.coef_ , index=X_data_n.columns )
            colname='alpha:'+str(param)
            coeff_df[colname] = coeff

    return coeff_df
# end of get_linear_regre_eval
```

alpha 값 리스트, 피처 데이터 세트, 타깃 데이터 세트

→ alpha 값에 따른 폴드 평균 RMSE, 회귀 계수값

규제 선형 모델

라쏘 회귀(Lasso)

✓ alpha 값에 따른 변화

```
#라쏘에 사용될 alpha 파라미터의 값을 정의하고 get_linear_reg_eval() 함수 호출
lasso_alphas = [ 0.07, 0.1, 0.5, 1, 3]
coeff_lasso_df = get_linear_reg_eval('Lasso', params=lasso_alphas, X_data_n=X_data, y_target_n=y_target)
```

```
# 반환된 coeff_lasso_df를 첫 번째 칼럼순으로 내림차순 정렬해 회귀계수 DataFrame을 출력
sort_column = 'alpha:'+str(lasso_alphas[0])
coeff_lasso_df.sort_values(by=sort_column, ascending=False)
```

```
#######  Lasso #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.612
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.615
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.669
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.776
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.189
```

규제 선형 모델

라쏘 회귀(Lasso)

✓ alpha 값에 따른 변화

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.789725	3.703202	2.498212	0.949811	0.000000
CHAS	1.434343	0.955190	0.000000	0.000000	0.000000
RAD	0.270936	0.274707	0.277451	0.264206	0.061864
ZN	0.049059	0.049211	0.049544	0.049165	0.037231
B	0.010248	0.010249	0.009469	0.008247	0.006510
NOX	-0.000000	-0.000000	-0.000000	-0.000000	0.000000
AGE	-0.011706	-0.010037	0.003604	0.020910	0.042495
TAX	-0.014290	-0.014570	-0.015442	-0.015212	-0.008602
INDUS	-0.042120	-0.036619	-0.005253	-0.000000	-0.000000
CRIM	-0.098193	-0.097894	-0.083289	-0.063437	-0.000000
LSTAT	-0.560431	-0.568769	-0.656290	-0.761115	-0.807679
PTRATIO	-0.765107	-0.770654	-0.758752	-0.722966	-0.265072
DIS	-1.176583	-1.160538	-0.936605	-0.668790	-0.000000

alpha의 크기가 증가함에 따라 일부 피처의 회귀계수값 = 0

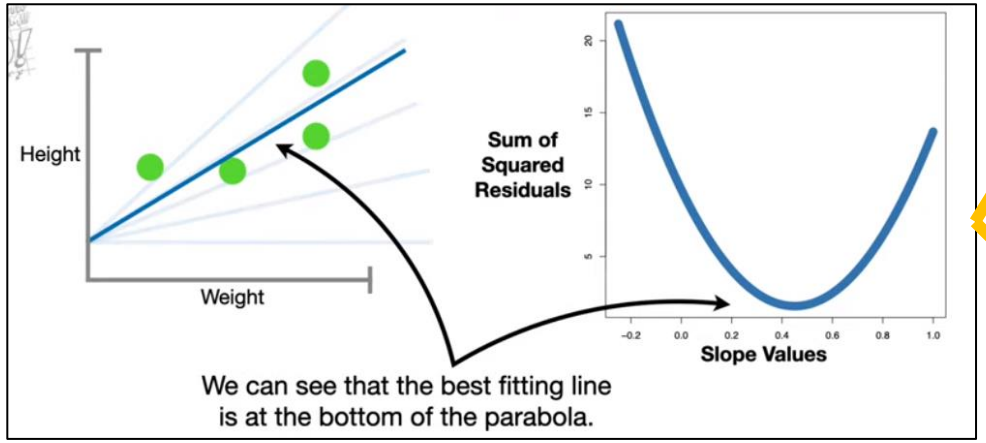
→ 피처 선택 → 모델 단순화 → 오버 피팅 ↓

But. 변수 선택에 따른 정보 손실 → 정확성 ↓

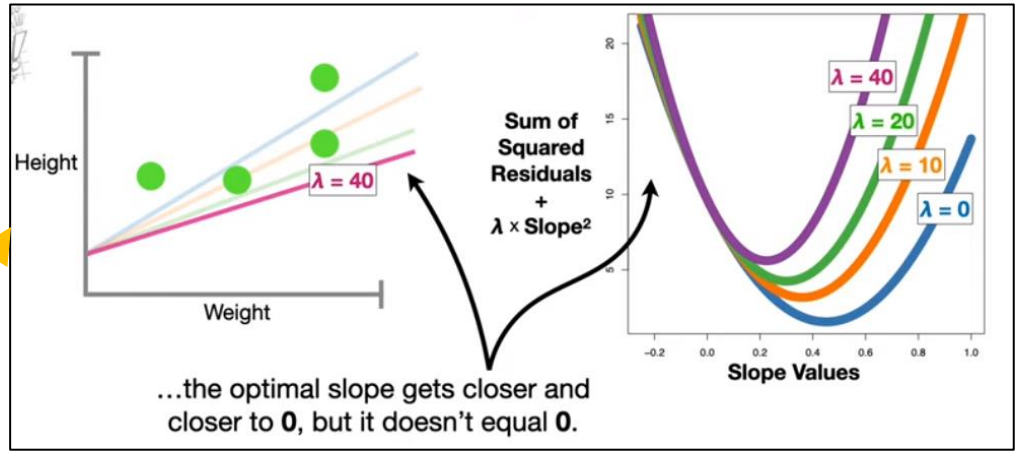
규제 선형 모델

릿지vs라쏘

몸무게에 따른 키 예측

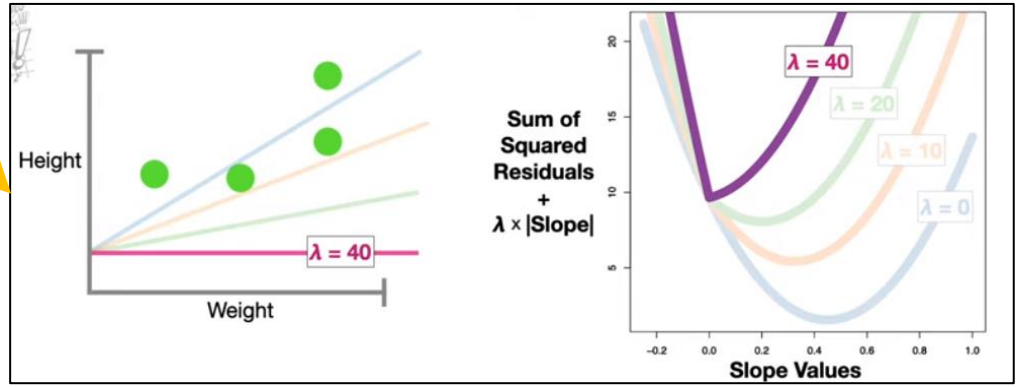


릿지



→ 변수 선택 X

라쏘



→ 변수 선택 O

→ 유의미한 변수가 적을 때는 라쏘가, 반대의 상황에서는 릿지의 성능이 더 좋음

규제 선형 모델

엘라스틱넷회귀(ElasticNet)

L2규제와 L1규제를 결합한 회귀

$$Min \quad RSS(W) + alpha1 * \|W\|_2^2 + alpha2 * \|W\|_1$$

✓ **라쏘 회귀**의 극단적 성향(중요피처 셀랙션, 회귀계수의 급격한 변동)을 완화 하기 위해 **L2 규제 추가**

➡ **수행시간**이 오래걸림

✓ EleasticNet 클래스 주요 파라미터

alpha	a*L1 + b*L2 (a=L1규제 alpha값, b=L2 규제 alpha값)
l1_ratio	a/(a+b) l1_ratio = 0 → a=0 → L2규제 l1_ratio = 1 → b=0 → L1규제

규제 선형 모델

엘라스틱넷회귀(ElasticNet)

✓ alpha 값에 따른 변화

```
# 엘라스틱넷에 사용될 alpha 파라미터의 값들을 정의하고 get_linear_reg_eval() 함수 호출
# l1_ratio는 0.7로 고정
elastic_alphas = [ 0.07, 0.1, 0.5, 1, 3]
coeff_elastic_df =get_linear_reg_eval('ElasticNet', params=elastic_alphas,
                                      X_data_n=X_data, y_target_n=y_target)
```

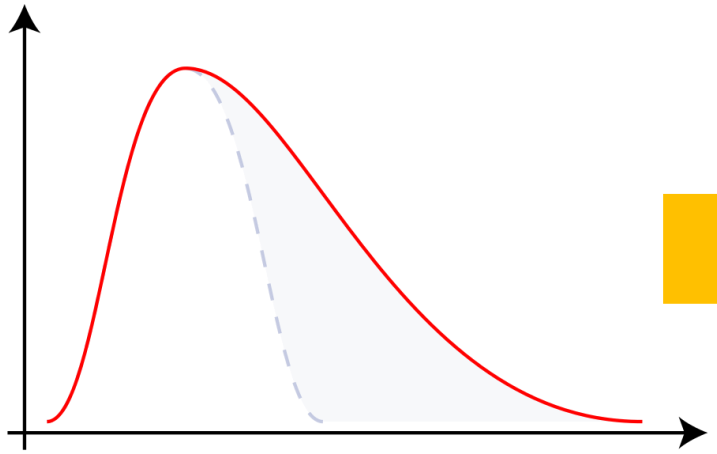
get_linear_get_eval( ) 함수 작성시 l1_ratio 고정

→ alpha 값에 따른 변화만 확인

```
#######  ElasticNet #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.542
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.526
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.467
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.597
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.068
```

규제 선형 모델

선형회귀 모델을 위한 데이터 변환



- 타깃값과 피처값의 정규분포가 아닌 왜곡된 분포는 예측 성능에 부정적 영향 (타깃값의 영향 > 피처값의 영향)

스케일링 /정규화

✓ StandardScaler
✓ MinMaxScaler

예측성능의 향상을 크게 기대하기 어려움

✓ 위를 수행한 데이터 세트에 다항 특성을 적용하여 변환
(위에서 예측성능 향상이 없을때 사용)

피처의 개수가 많을 경우,
피처 개수의 기하급수적 증가 → 과적합 우려

✓ **Log 변환**

많은 사례에서 예측 성능 향상

규제 선형 모델

선형회귀 모델을 위한 데이터 변환

✓ 변환 방법별 예측 성능 비교

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

# method는 표준 정규 분포 변환(Standard), 최대값/최소값 정규화(MinMax), 로그변환(Log) 결정
# p_degree는 다항식 특성을 추가할 때 적용. p_degree는 2이상 부여하지 않음.
def get_scaled_data(method='None', p_degree=None, input_data=None):
    if method == 'Standard':
        scaled_data = StandardScaler().fit_transform(input_data)
    elif method == 'MinMax':
        scaled_data = MinMaxScaler().fit_transform(input_data)
    elif method == 'Log':
        scaled_data = np.log1p(input_data)
    else:
        scaled_data = input_data

    if p_degree != None:
        scaled_data = PolynomialFeatures(degree=p_degree,
                                         include_bias=False).fit_transform(scaled_data)

    return scaled_data
```

데이터 변환 함수 get_scaled_data( ) 정의

- method = 표준정규분포 변환(Standard),
  최댓값/최솟값 정규화(MinMax)
  로그변환(Log)
- P_degree : 다항식 특성 추가시 다항식 차수

규제 선형 모델

선형회귀 모델을 위한 데이터 변환

✓ 변환 방법별 예측 성능 비교

```
# Ridge의 alpha값을 다르게 적용하고 다양한 데이터 변환방법에 따른 RMSE 추출.
alphas = [0.1, 1, 10, 100]
#변환 방법은 모두 6개, 원본 그대로, 표준정규분포, 표준정규분포+다항식 특성
# 최대/최소 정규화, 최대/최소 정규화+다항식 특성, 로그변환
scale_methods=[(None, None), ('Standard', None), ('Standard', 2),
               ('MinMax', None), ('MinMax', 2), ('Log', None)]
for scale_method in scale_methods:
    X_data_scaled = get_scaled_data(method=scale_method[0], p_degree=scale_method[1],
                                    input_data=X_data)
    print(X_data_scaled.shape, X_data.shape)
    print('₩n## 변환 유형:{0}, Polynomial Degree:{1}'.format(scale_method[0], scale_method[1]))
    get_linear_reg_eval('Ridge', params=alphas, X_data_n=X_data_scaled,
                        y_target_n=y_target, verbose=False, return_coeff=False)
```

- 변환 X
- 표준정규분포
- 표준정규분포 - 2차다항식 변환
- 최솟값/최댓값 정규화
- 최솟값/최댓값 정규화 - 2차 다항식 변환
- 로그변환

규제 선형 모델

선형회귀 모델을 위한 데이터 변환

✓ 변환 방법별 예측 성능 비교

변환유형	alpha값			
	alpha=0.1	alpha=1	alpha=10	alpha=100
원본 데이터	5.796	5.659	5.524	5.332
표준 정규 분포	5.834	5.810	5.643	5.424
표준 정규 + 2차 다항식	8.776	6.849	5.487	**4.631**
최솟값/최댓값	5.770	5.468	5.755	7.635
최솟값/최댓값 + 2차 다항식	5.294	**4.320**	5.186	6.538
로그변환	**4.772**	**4.676**	**4.835**	6.244

일차 변환 후 정규 변환시 성능 개선

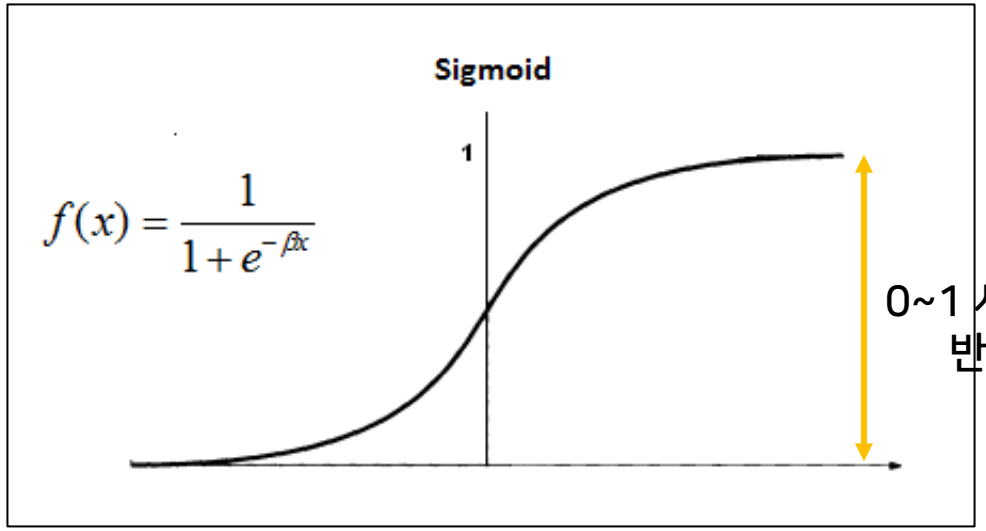
But. 다항식 변환은 데이터 증가에 따라 계산 시간이 많이 소모됨.

로그변환의 성능 우수

2. 로지스틱 회귀

로지스틱 회귀

로지스틱 회귀



- ✓ **선형 회귀** 방식을 **분류**에 적용한 알고리즘
- ✓ **시그모이드(Sigmoid) 함수** 최적선을 찾고 이 함수의 반환 값을 **확률(0~1)**로 간주, 그 확률에 따라 **더 가능성이 높은 범주로 분류(이진 분류)**
- ✓ 많은 자연, 사회 현상에서 특정 변수의 확률 값은 선형이 아닌 S자 커브의 형태를 가짐

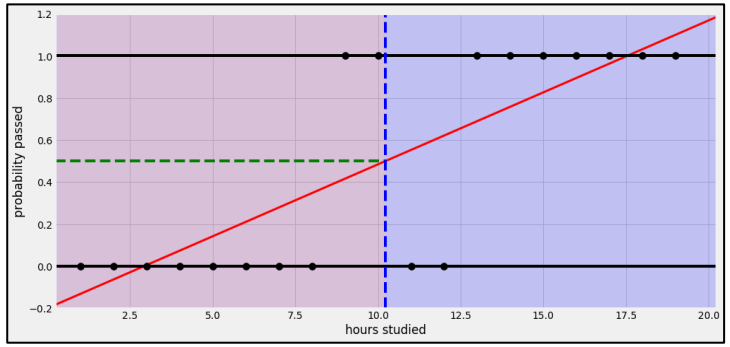
로지스틱 함수

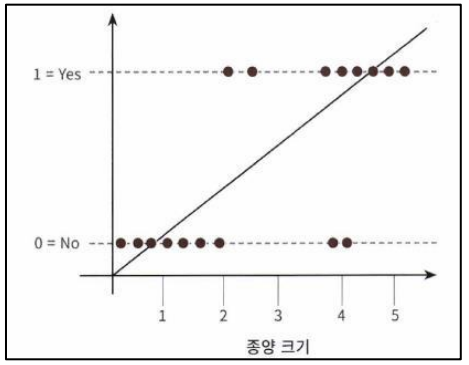
그림 출처 : https://wikidocs.net/32046

로지스틱 회귀

선형 vs 로지스틱

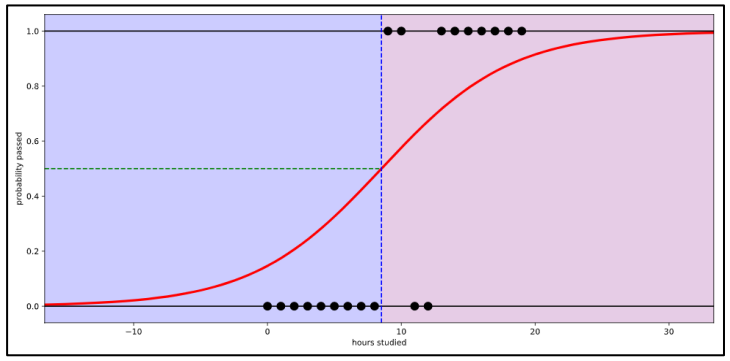
✓ 선형 회귀

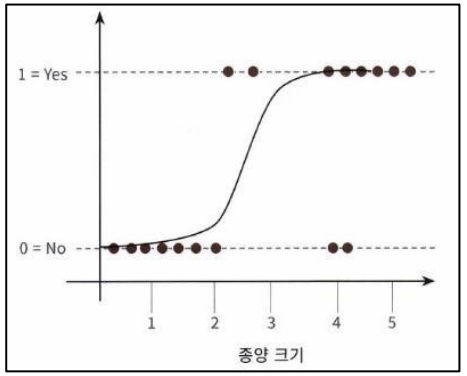




- 음과 양의 방향으로 무한대의 확률
- 분류의 정확도 ↓

✓ 로지스틱 회귀





- 0~1 사이의 확률
- 분류의 정확도 ↑

Ex) 학습시간에 따른 시험합격 확률

Ex) 종양 크기에 따른 악성 종양 여부

로지스틱 회귀

로지스틱 확률 예측 과정

1. 모든 피처들의 계수(coefficient)와 절편(intercept)을 0으로 초기화한다
2. 각 속성들의 값(value)에 계수(coefficient)를 곱해서 **log-odds**를 구한다.
3. **log-odds**를 **sigmoid 함수**에 넣어서 [0,1] 범위의 확률을 구한다.

로지스틱 회귀

Log-odds(로짓)

$w_o + w_1 * x_1 + w_2 * x_2 + \cdots$
$= W \cdot X$ → $y(0\ or\ 1)$

↓

y=1일 **확률(Probability)** 예측 → Probability의 범위 [0,1]
If) y(probability) = x-4 일 때,
x=10이면 →범위를 벗어남

↓

Odds(오즈) 예측 → Odds의 범위 [0, ∞)

↓

Log-odds(로짓) 예측 → Log-odds 의 범위 (- ∞, ∞)

Odds(오즈) 특정결과의 가능성을 측정

$$\frac{p}{1-\mathrm{p}}$$

ex) 게임에서 이길 오즈가 1:4(1/4)
→ 이길확률 :1/5, 질 확률 4/5

Log-odds(로짓)

$$\log(\frac{p}{1-\mathrm{p}})$$

로지스틱 회귀

로지스틱 확률 예측 과정

Log-odds(로짓) $\log_2 \frac{p}{1-p} = \ln(\frac{p}{1-p})$

$$z(\text{로짓}) = \ln\left(\frac{p}{1-p}\right) = w_o + w_1 * x_1 + w_2 * x_2 + \cdots$$

$$= W \cdot X$$

$$p = \frac{1}{1+e^{-W \cdot X}} = \frac{1}{1+e^{-z}}$$

로지스틱 함수

$\frac{p}{1-p} = e^{W \cdot X}$

양 변에 역수를 취하자

$\frac{1-p}{p} = \frac{1}{p} - 1 = \frac{1}{e^{W \cdot X}}$

$\frac{1}{p} = \frac{1}{e^{W \cdot X}} + 1 = \frac{1}{e^{W \cdot X}} + \frac{e^{W \cdot X}}{e^{W \cdot X}}$

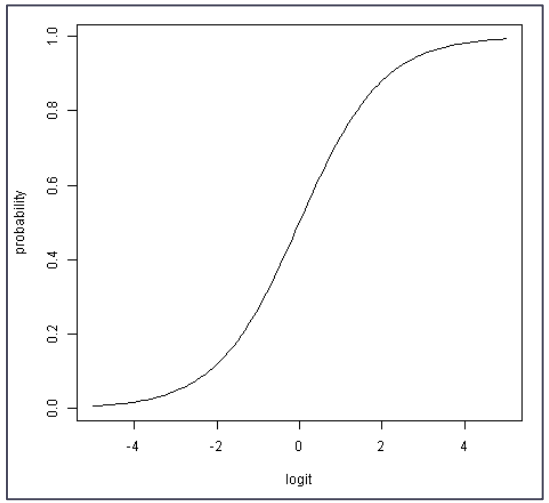
$= \frac{1+e^{W \cdot X}}{e^{W \cdot X}}$

다시 한 번 역수를 취하자

$p = \frac{e^{W \cdot X}}{1+e^{W \cdot X}}$

여기서 우리가 아는 식으로 한번 더 변형하면

$p = \frac{e^{W \cdot X}}{1+e^{W \cdot X}} \cdot \frac{\frac{1}{e^{W \cdot X}}}{\frac{1}{e^{W \cdot X}}} = \frac{1}{\frac{1}{e^{W \cdot X}}+1} = \frac{1}{1+e^{-W \cdot X}}$



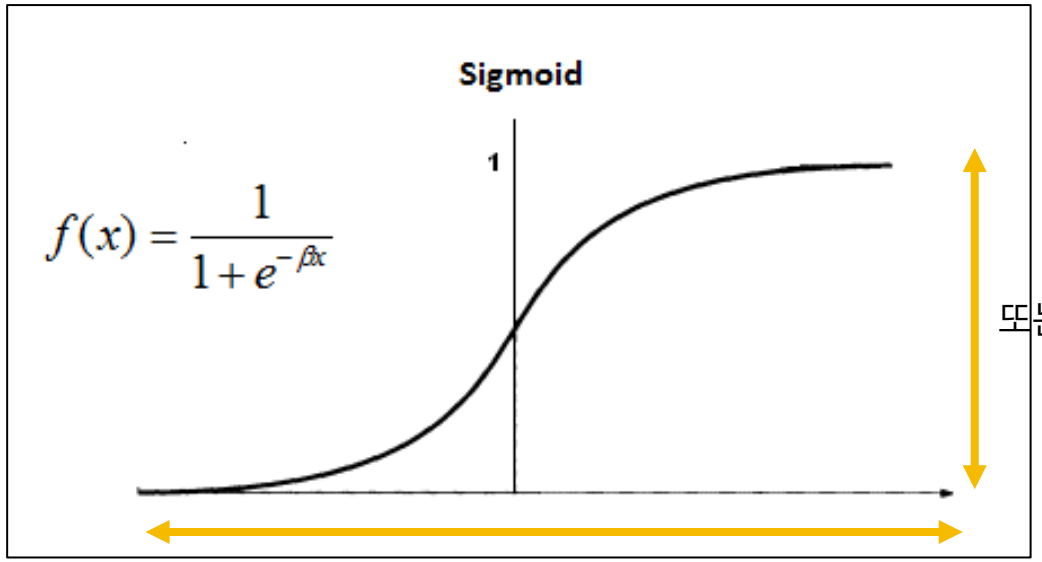
확률에 기반하여 0,1 분류

출처 : https://mazdah.tistory.com/769, https://haje01.github.io/2019/11/19/logit.html

로지스틱 회귀

시그모이드(Sigmoid) 함수

S자형 곡선 혹은 시그모이드 곡선을 갖는 함수



종류

✓ 로지스틱(logistic)-대표적

$$f(x) = \frac{1}{1+e^{-x}} \quad (0\sim1)$$

✓ 아크탄젠트

$$f(x) = \arctan x \quad (-1\sim1)$$

✓ 오차 함수

$$f(x) = \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt \quad (-1\sim1)$$

로지스틱 회귀

로지스틱 결과 해석-Odds Ratio

교차비(Odds Ratio)

$$Odds\ Ratio = \frac{Odds1}{Odds2}$$

질병발생 / 요인노출	예	아니오	합계
예	a	b	a+b
아니오	c	d	c+d
합계	a+c	b+d	n=a+b+c+d

* 위험요인 노출에 따라 질병이 발생할 교차비(Odds ratio)

1. 요인에 노출된 집단에서 환자가 나올 오즈

$$\frac{\text{요인에 노출된 집단에서 환자가 나올 확률}}{\text{요인에 노출된 집단에서 환자가 나오지 않을 확률}} = \frac{\frac{a}{a+b}}{\frac{b}{a+b}} = \frac{a}{b}$$

2. 요인에 노출되지 않은 집단에서 환자가 나올 오즈

$$\frac{\text{요인에 노출되지 않은 집단에서 환자가 나올 확률}}{\text{요인에 노출되지 않은 집단에서 환자가 나오지 않을 확률}} = \frac{\frac{c}{c+d}}{\frac{d}{c+d}} = \frac{c}{d}$$

3. 교차비(Odds ratio)

$$\frac{\text{요인에 노출된 집단에서 환자가 나올 오즈}}{\text{요인에 노출되지 않은 집단에서 환자가 나올 오즈}} = \frac{\frac{a}{b}}{\frac{c}{d}} = \frac{ad}{bc}$$

로지스틱 회귀

로지스틱 결과 해석-Odds Ratio

	질병 발생 확률	질병 미발생 확률
요인에 노출(x_1=1)	p	1-p
요인에 노출 X(x_1=0)	p′	1-p′

$$z(\text{로짓}) = \ln\left(\frac{p}{1-p}\right) = w_o + w_1 * x_1 + w_2 * x_2 + \cdots$$

$$= W \cdot X$$

If) x_1이 요인 노출 여부

1. z_1 = 식1(x_1=1)
2. z_2 = 식2(x_1=0)
3. $z_2 - z_1 = w_1(1 - 0)$

$$\ln\left(\frac{p}{1-p}\right) - \ln\left(\frac{p'}{1-p'}\right) = w_1$$

4. ln(노출 O 오즈/노출 X 오즈) = w_1
5. ln(오즈비) = w_1
6. 오즈비=exp(w_1)

로지스틱 회귀

로지스틱 결과 해석-Odds Ratio

$$Odds\ Ratio = \frac{Odds1}{Odds2}$$

$Odds1$ → 요인에 노출된 집단에서 환자가 나올 오즈

$Odds2$ → 요인에 노출되지 않은 집단에서 환자가 나올 오즈

1. OR=1

→ Odds1=Odds2

→ 위험 요인에 대한 노출이 질병 발생에 유의미한 영향 X

2. OR>1

→ 위험 요인에 노출 되었을 때 질병이 발생할 오즈 ↑

3. OR<1

→ 위험 요인에 노출 되었을 때 질병이 발생할 오즈 ↓

만약,

OR=1.5

→ 위험요인에 노출 될 경우(아닐 경우에 비해) 환자가 나올 가능성 1.5배

→ 환자가 나올 가능성 50% 증가

OR=0.8

→ 위험요인에 노출 될 경우(아닐 경우에 비해) 환자가 나올 확률 0.8배

→ (0.8-1)*100= -20%, 환자가 나올 가능성 20% 감소

로지스틱 회귀

LogLoss

$$LogLoss = -\frac{1}{n}\sum_{i=0}^{n}[y_i log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

각 확률값에 음의 로그를 취한 후 모두 합하여 1/n

→ 최종 Logloss

Ex) 100%의 확률로 예측한 경우 → Logloss = -log(1.0)=0

80% 확률로 예측한 경우 → Logloss = -log(0.8)=0.22314

60% 확률로 예측한 경우 → Logloss = -log(0.6) = 0.51082

최종적으로 맞춘 결과만 가지고 성능을 평가할 경우,
얼만큼의 확률로 해당 답을 얻었는지 평가 불가.

- ✓ 분류모델 평가 시 사용
- ✓ 모델이 예측한 확률 값을 직접적으로 반영
- ✓ 음의 값 → 낮은 확률에 대한 패널티

로지스틱 회귀

✓ 로지스틱 분류 실습 – 위스콘신 유방암 데이터

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression

cancer = load_breast_cancer()
```

선형 회귀 계열의 로지스틱 회귀는 데이터 정규 분포도에 따라 예측 성능 영향을 받을 수 있음

→ 정규 분포 형태의 표준 스케일링

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# StandardScaler( )로 평균이 0, 분산 1로 데이터 분포도 변환
scaler = StandardScaler()
data_scaled = scaler.fit_transform(cancer.data)

X_train , X_test, y_train , y_test = train_test_split(data_scaled, cancer.target, test_size=0.3, random_state=0)
```

로지스틱 회귀

로지스틱 회귀

```
from sklearn.metrics import accuracy_score, roc_auc_score

# 로지스틱 회귀를 이용하여 학습 및 예측 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_preds = lr_clf.predict(X_test)

# accuracy측정
print('accuracy: {:0.3f}'.format(accuracy_score(y_test, lr_preds)))
```

```
accuracy: 0.977
```

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*
Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

dual : *bool, default=False*
Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

tol : *float, default=1e-4*
Tolerance for stopping criteria.

C : *float, default=1.0*
Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

주요 파라미터

penalty : 규제 유형('l2' = L2규제, 'l1' = L1규제,
디폴트 = 'l2')

C : 규제 강도(alpha의 역수)
C값이 작을수록 규제 강도가 큼

로지스틱 회귀

로지스틱 회귀

✓ 하이퍼 파라미터 최적화

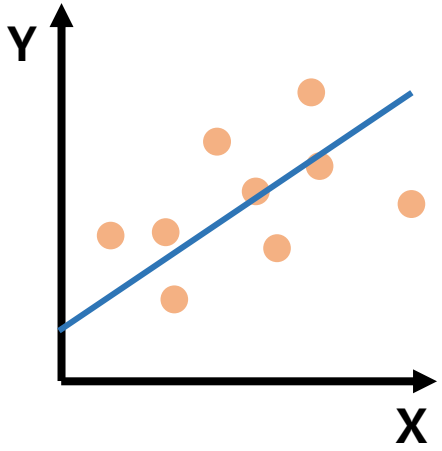
```
from sklearn.model_selection import GridSearchCV

params={'penalty':['l2', 'l1'],
        'C':[0.01, 0.1, 1, 1, 5, 10]}

grid_clf = GridSearchCV(lr_clf, param_grid=params, scoring='accuracy', cv=3 )
grid_clf.fit(data_scaled, cancer.target)
print('최적 하이퍼 파라미터:{0}, 최적 평균 정확도:{1:.3f}'.format(grid_clf.best_params_,
                                                    grid_clf.best_score_))
```

```
최적 하이퍼 파라미터:{'C': 1, 'penalty': 'l2'}, 최적 평균 정확도:0.975
```

3. 회귀 트리

회귀 트리

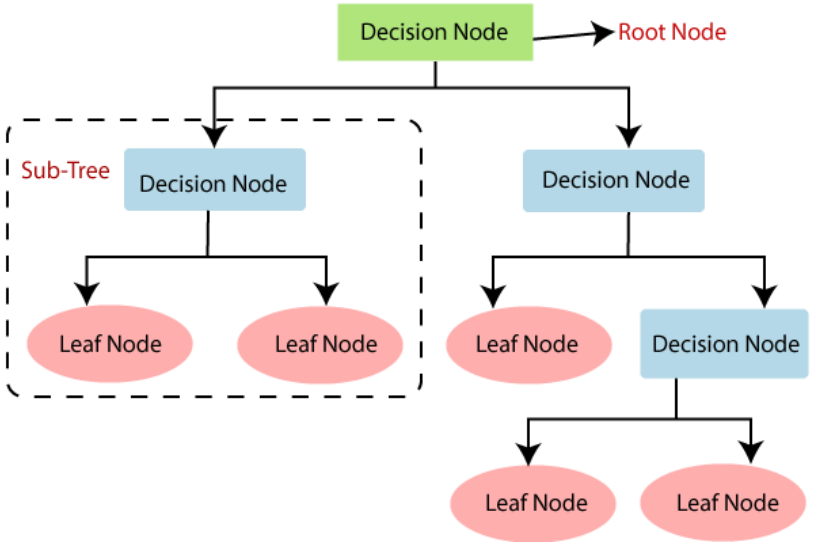
선형회귀 vs 회귀 트리



$$f(x) = w_0 + w_1 * x$$

선형 회귀

→ 회귀 계수 기반 최적 회귀 함수 도출



분류트리와 유사

회귀 트리

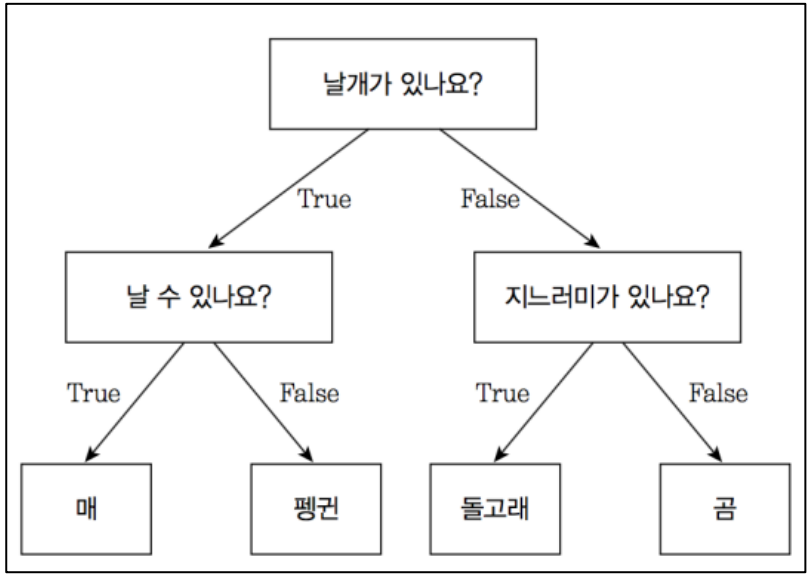
→ 트리기반 회귀방식

→ **리프노드에 속한 데이터 의 평균값**을 구해 회귀 예측값 계산

그림 출처 : https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

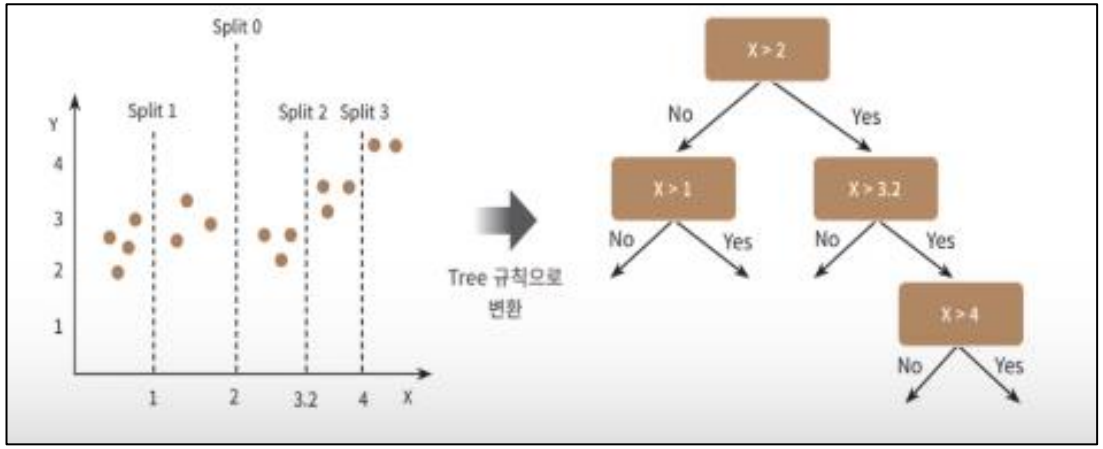
회귀 트리

분류트리 vs 회귀 트리



분류트리

→ 리프노드에서 특정 클래스 레이블을 결정



트리 기반 회귀 방식

→ 리프노드에서 회귀 예측값을 계산

출처 : https://roytravel.tistory.com/63

회귀 트리

회귀 트리의 장단점

장점

- ✓ 의사결정 트리의 장점과 수치 데이터를 모델링하는 능력의 결합
- ✓ 사용자가 모델을 미리 명시하지 않아도됨
- ✓ 자동 특징 선택 사용
- ✓ 일부 데이터 타입에 대해 선형회귀보다 아주 잘 맞음
- ✓ 모델 해석에 통계지식이 필요하지 않음

단점

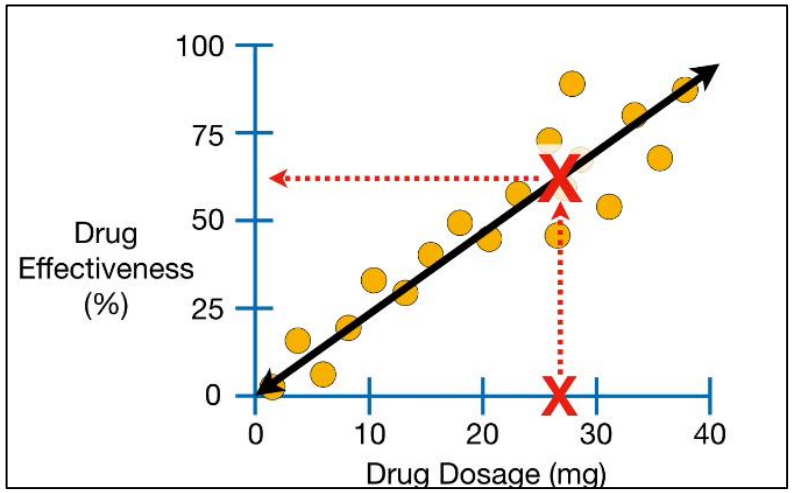
- ✓ 선형 회귀만큼 잘 알려져 있지 않음
- ✓ 많은 양의 훈련 데이터 필요
- ✓ 결과에 대한 개별 특징의 전체적인 순 영향을 알기 어려움
- ✓ 큰 트리는 회귀 모델보다 해석하기 조금 어려움
- ✓ 변수가 많을 경우 시각화 어려움

출처 : https://osh88itopia.tistory.com/142

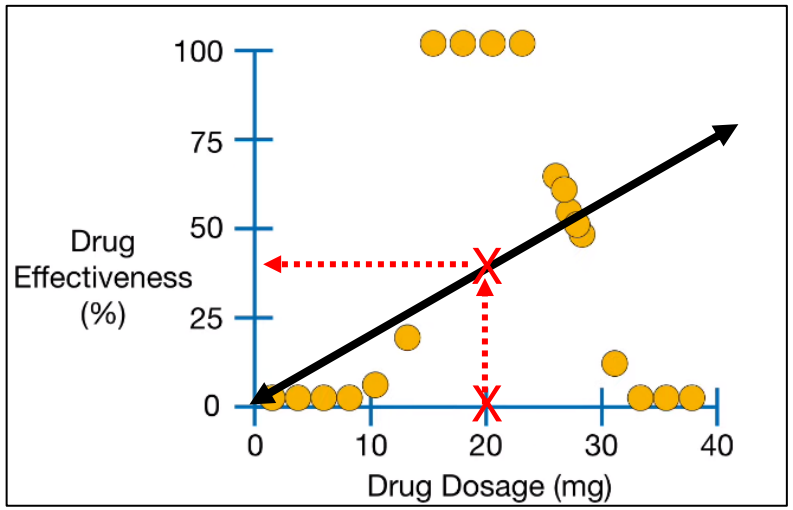
회귀 트리

회귀 트리 예측 과정

✓ 약 복용량에 따른 약 복용 효과 예측  vs. 



데이터가 선형성일 때 잘 예측

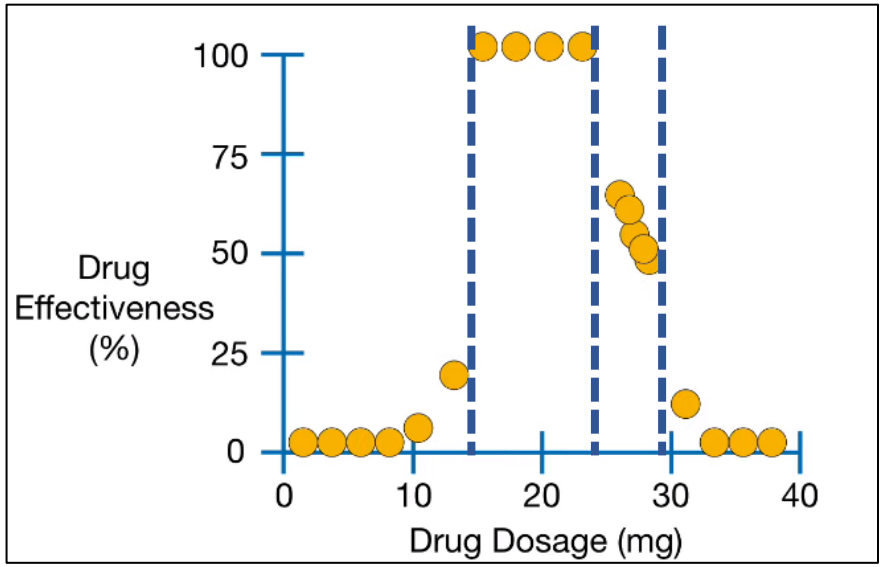


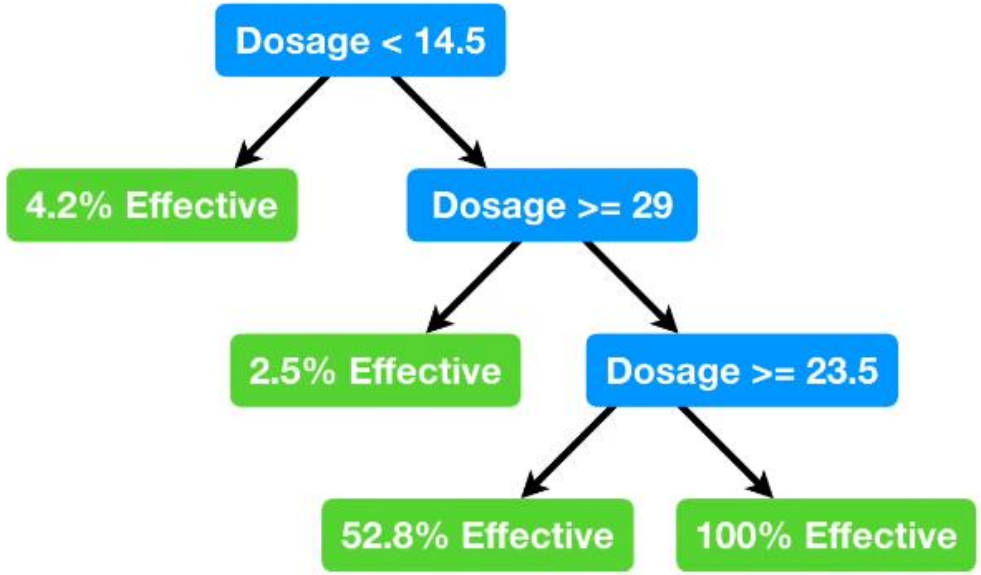
데이터가 선형이 아닐 때 부정확

회귀 트리

회귀 트리 예측 과정

✓ 약 복용량에 따른 약 복용 효과 예측



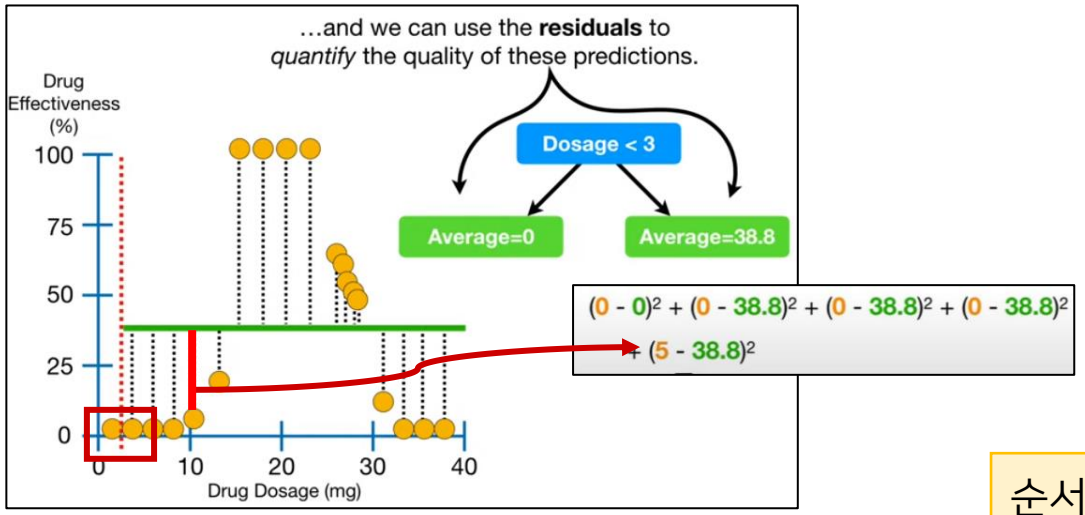


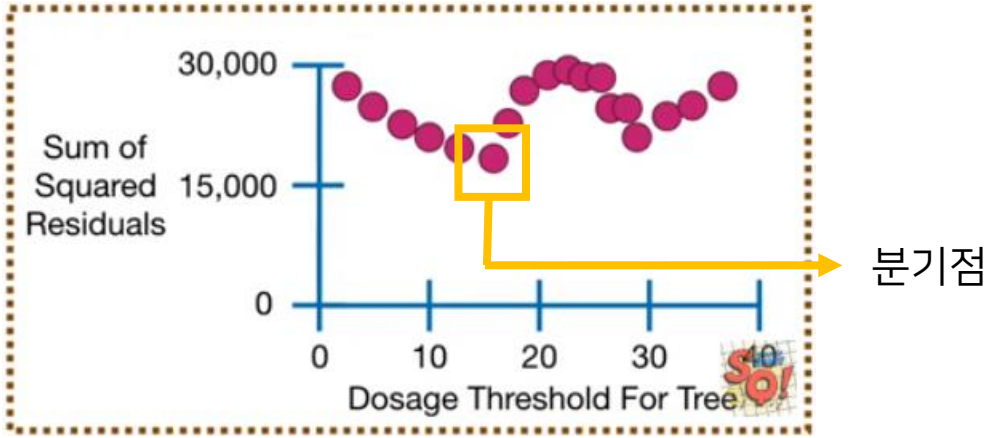
RSS를 최소화 하는 규칙 기준에 따라 분할

회귀 트리

회귀 트리 예측 과정

✓ 약 복용량에 따른 약 복용 효과 예측





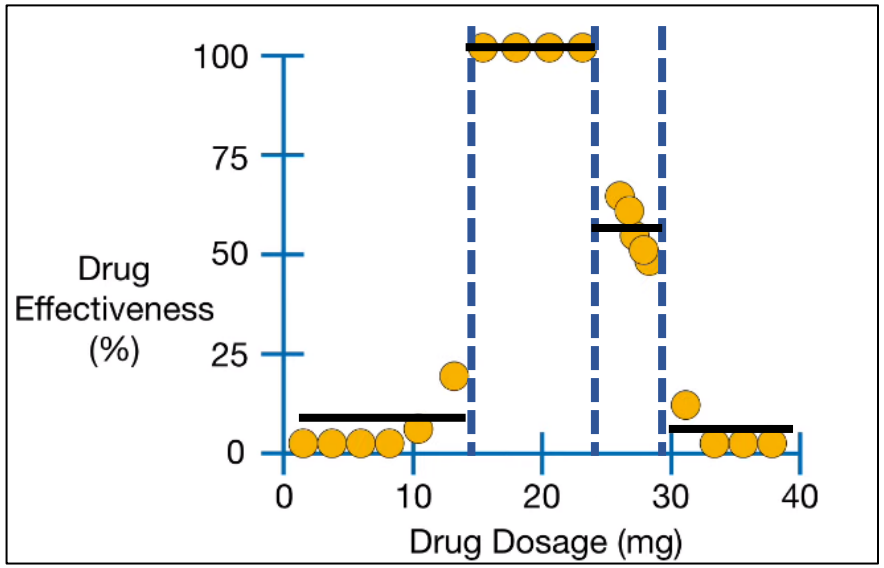
분기점

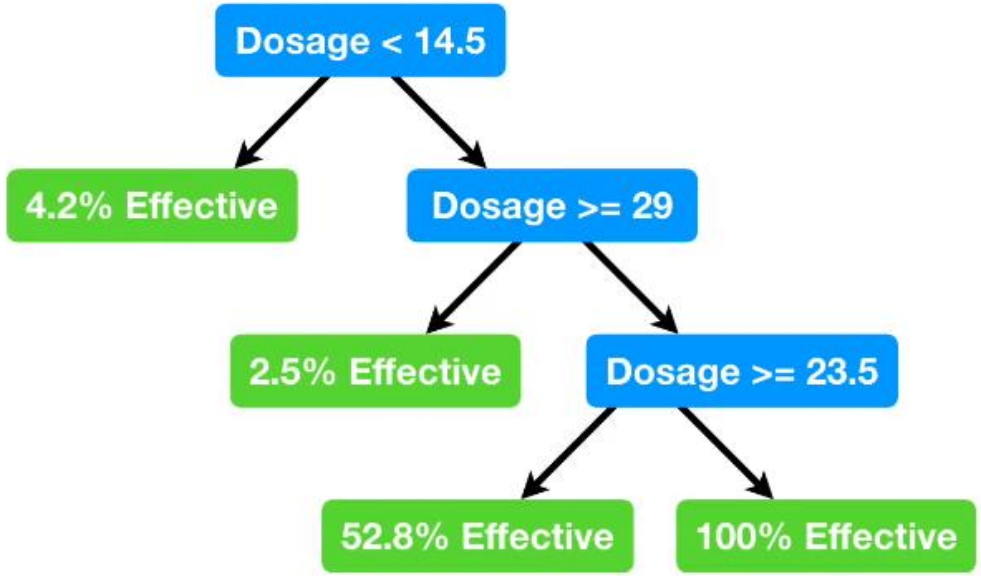
순서대로 **연속한 두 변수의 평균값**을 임계값으로 설정하여
그에 따른 잔차제곱합(RSS)을 비교하여 최소값을 갖는 임계값을
분기점으로 설정

회귀 트리

회귀 트리 예측 과정

✓ 약 복용량에 따른 약 복용 효과 예측

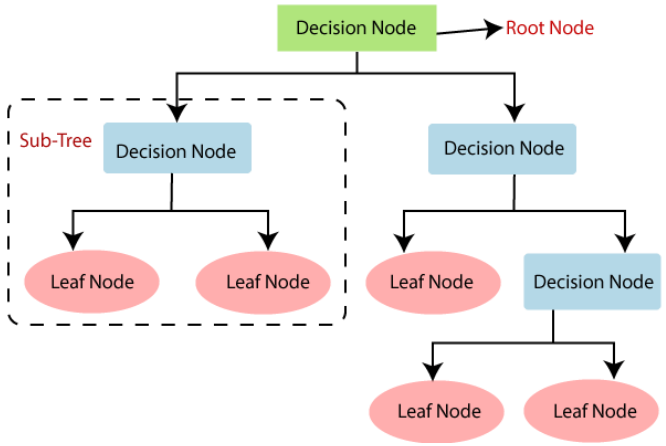




리프노드에 소속된 데이터 값의 평균값 → 최종 리프 노드의 결정값

회귀 트리

회귀 트리



- ✓ 분류트리와 마찬가지로 깊이가 깊어질 수록 과적합
- ✓ 회귀 트리의 하이퍼 파라미터는 분류트리와 유사

class `sklearn.ensemble.` **`RandomForestClassifier`**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) ¶ [source]

class `sklearn.ensemble.` **`RandomForestRegressor`**(*n_estimators=100, *, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

회귀 트리

알고리즘	회귀 Estimator 클래스	분류 Estimator 클래스
Decision Tree	DecisionTreeRegressor	DecisionTreeClassifier
Gradient Boosting	GradientBoostingRegressor	GradientBoostingClassifier
XGBoost	XGBRegressor	XGBClassifier
LightGBM	LGBMRegressor	LGBMClassifier

- ✓ 트리 기반 알고리즘은 분류뿐만 아니라 회귀도 가능
- ✓ 트리 생성은 CART(Classification And **Regression** Trees) 알고리즘 기반

회귀 트리

✓ 랜덤 포레스트 회귀 트리 – 보스턴 주택 가격 예측 실습

```
from sklearn.datasets import load_boston
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np

# 보스턴 데이터 세트 로드
boston = load_boston()
bostonDF = pd.DataFrame(boston.data, columns = boston.feature_names)

bostonDF['PRICE'] = boston.target
y_target = bostonDF['PRICE']
X_data = bostonDF.drop(['PRICE'], axis=1,inplace=False)

rf = RandomForestRegressor(random_state=0, n_estimators=1000)
neg_mse_scores = cross_val_score(rf, X_data, y_target, scoring="neg_mean_squared_error", cv = 5)
rmse_scores  = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print(' 5 교차 검증의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 2))
print(' 5 교차 검증의 개별 RMSE scores : ', np.round(rmse_scores, 2))
print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

```
5 교차 검증의 개별 Negative MSE scores:  [ -7.88 -13.14 -20.57 -46.23 -18.88]
5 교차 검증의 개별 RMSE scores :  [2.81 3.63 4.54 6.8  4.34]
5 교차 검증의 평균 RMSE : 4.423
```

회귀 트리

회귀 트리

✓ 다양한 회귀 트리 실습

```
def get_model_cv_prediction(model, X_data, y_target):
    neg_mse_scores = cross_val_score(model, X_data, y_target, scoring="neg_mean_squared_error", cv = 5)
    rmse_scores  = np.sqrt(-1 * neg_mse_scores)
    avg_rmse = np.mean(rmse_scores)
    print('##### ',model.__class__.__name__ , ' #####')
    print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

모델에 따라 교차검증 평균 RMSE를 계산하는 함수 get_model_cv_prediction( )정의

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

dt_reg = DecisionTreeRegressor(random_state=0, max_depth=4)
rf_reg = RandomForestRegressor(random_state=0, n_estimators=1000)
gb_reg = GradientBoostingRegressor(random_state=0, n_estimators=1000)
xgb_reg = XGBRegressor(n_estimators=1000)
lgb_reg = LGBMRegressor(n_estimators=1000)

# 트리 기반의 회귀 모델을 반복하면서 평가 수행
models = [dt_reg, rf_reg, gb_reg, xgb_reg, lgb_reg]
for model in models:
    get_model_cv_prediction(model, X_data, y_target)
```

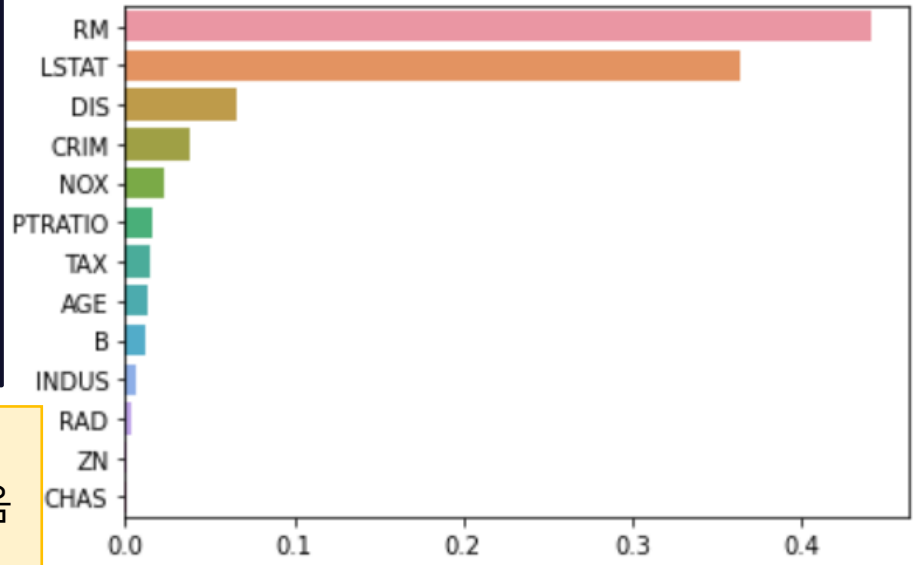
```
#####  DecisionTreeRegressor  #####
 5 교차 검증의 평균 RMSE : 5.978
#####  RandomForestRegressor  #####
 5 교차 검증의 평균 RMSE : 4.423
#####  GradientBoostingRegressor  #####
 5 교차 검증의 평균 RMSE : 4.269
#####  XGBRegressor  #####
 5 교차 검증의 평균 RMSE : 4.251
#####  LGBMRegressor  #####
 5 교차 검증의 평균 RMSE : 4.646
```

회귀 트리

회귀 트리

✓ 피처별 중요도 확인

```
import seaborn as sns
%matplotlib inline

rf_reg = RandomForestRegressor(n_estimators=1000)

# 앞 예제에서 만들어진 X_data, y_target 데이터 셋을 적용하여 학습합니다.
rf_reg.fit(X_data, y_target)

feature_series = pd.Series(data=rf_reg.feature_importances_, index=X_data.columns )
feature_series = feature_series.sort_values(ascending=False)
sns.barplot(x= feature_series, y=feature_series.index)
```



※ 회귀 트리 Regressor 클래스는 회귀 계수를 제공하는 coeff_ 속성이 없음

회귀 트리의 하이퍼 파라미터는 분류트리와 거의 동일

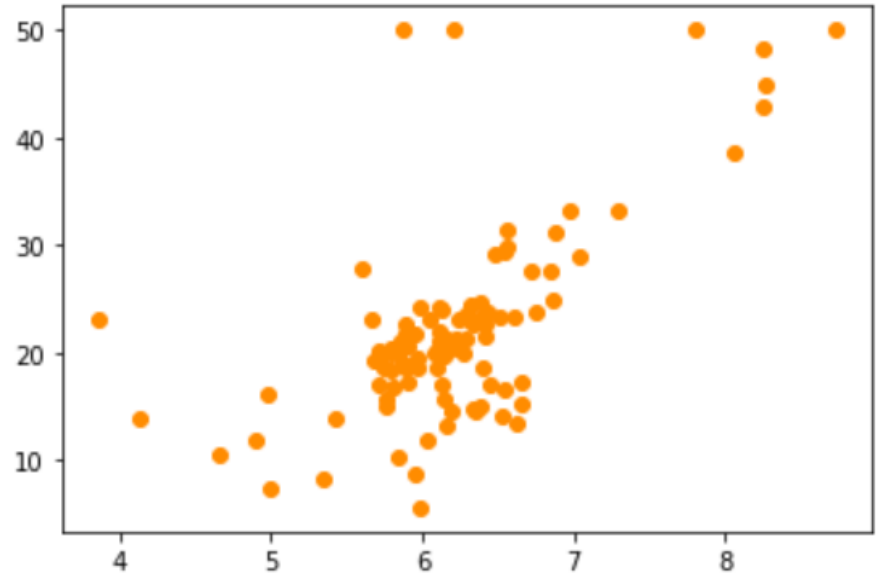
회귀 트리

회귀 트리

✓ 회귀 트리의 예측방법

2차원 평면상 표현을 위한 변수 추출

```
import matplotlib.pyplot as plt
%matplotlib inline

bostonDF_sample = bostonDF[['RM','PRICE']]
bostonDF_sample = bostonDF_sample.sample(n=100,random_state=0)
print(bostonDF_sample.shape)
plt.figure()
plt.scatter(bostonDF_sample.RM , bostonDF_sample.PRICE,c="darkorange")
```



결정 트리의 하이퍼 파라미터인 max_depth의 크기를 변화시키면서

회귀 트리의 예측선 변화 확인

회귀 트리

```
import numpy as np
from sklearn.linear_model import LinearRegression

# 선형 회귀와 결정 트리 기반의 Regressor 생성. DecisionTreeRegressor의 max_depth는 각각 2, 7
lr_reg = LinearRegression()
rf_reg2 = DecisionTreeRegressor(max_depth=2)
rf_reg7 = DecisionTreeRegressor(max_depth=7)

# 실제 예측을 적용할 테스트용 데이터 셋을 4.5 ~ 8.5 까지 100개 데이터 셋 생성.
X_test = np.arange(4.5, 8.5, 0.04).reshape(-1, 1)

# 보스턴 주택가격 데이터에서 시각화를 위해 피처는 RM만, 그리고 결정 데이터인 PRICE 추출
X_feature = bostonDF_sample['RM'].values.reshape(-1,1)
y_target = bostonDF_sample['PRICE'].values.reshape(-1,1)

# 학습과 예측 수행.
lr_reg.fit(X_feature, y_target)
rf_reg2.fit(X_feature, y_target)
rf_reg7.fit(X_feature, y_target)

pred_lr = lr_reg.predict(X_test)
pred_rf2 = rf_reg2.predict(X_test)
pred_rf7 = rf_reg7.predict(X_test)
```

LinearRegression과 DecisionTreeRegressor

의 max_depth를 각각 2,7로 해서 학습

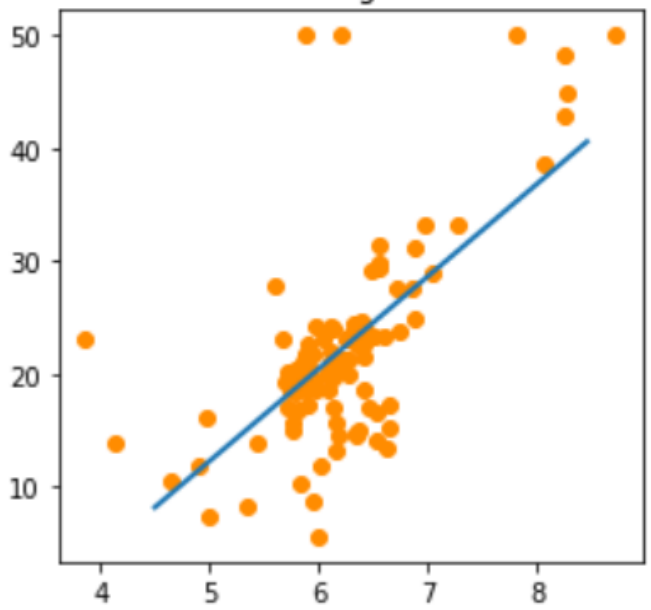
회귀 트리

```
fig , (ax1, ax2, ax3) = plt.subplots(figsize=(14,4), ncols=3)

# X축값을 4.5 ~ 8.5로 변환하며 입력했을 때, 선형 회귀와 결정 트리 회귀 예측 선 시각화
# 선형 회귀로 학습된 모델 회귀 예측선
ax1.set_title('Linear Regression')
ax1.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax1.plot(X_test, pred_lr,label="linear", linewidth=2 )

# DecisionTreeRegressor의 max_depth를 2로 했을 때 회귀 예측선
ax2.set_title('Decision Tree Regression: ₩n max_depth=2')
ax2.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax2.plot(X_test, pred_rf2, label="max_depth:3", linewidth=2 )

# DecisionTreeRegressor의 max_depth를 7로 했을 때 회귀 예측선
ax3.set_title('Decision Tree Regression: ₩n max_depth=7')
ax3.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax3.plot(X_test, pred_rf7, label="max_depth:7", linewidth=2)
```

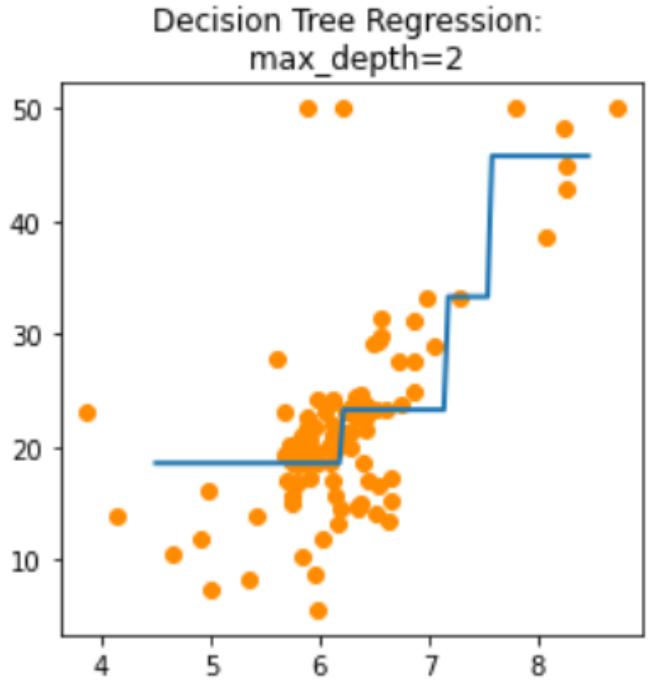
그래프 그리기

회귀 트리

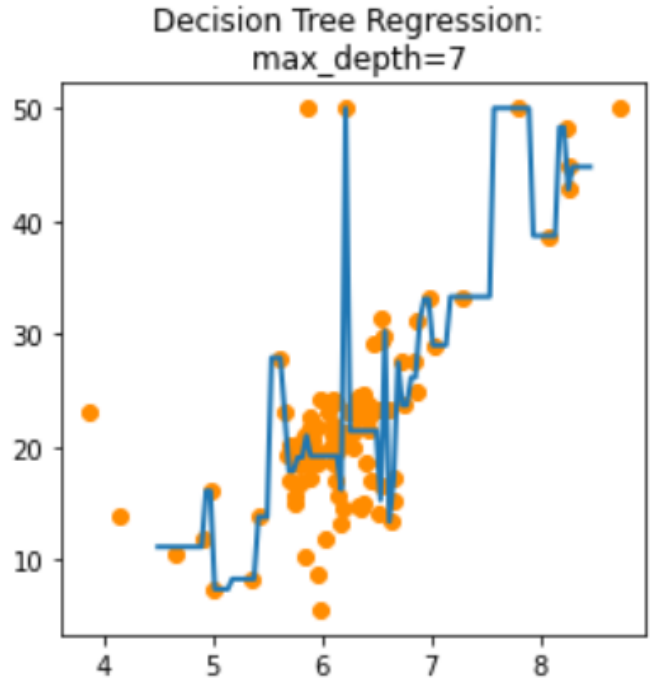
회귀 트리



선형회귀 : 직선 예측 회귀선



회귀 트리 : 분할데이터지점에서 계단형태



과적합이 되기 쉬운모델

4. 회귀 실습

자전거 대여 수요 예측

회귀 실습 – 자전거 대여 수요 예측

캐글의 자전거 대여 수요(Bike Sharing Demand) 예측 경연에서 사용된 학습 데이터 세트
https://www.kaggle.com/c/bike-sharing-demand/data

2011년 1월부터 2012년 12월까지 1시간 간격 동안의 자전거 대여 횟수

주요 칼럼

datetime: hourly date + timestamp

season: 1=봄, 2=여름, 3=가을, 4=겨울

holiday: 1=주말을 제외한 국경일 등의 휴일, 0=휴일이 아닌 날

workingday: 1=주말 및 휴일이 아닌 주중, 0=주말 및 휴일

weather:

1=맑음, 약간 구름 낀 흐림

2=안개, 안개 + 흐림

3=가벼운 눈, 가벼운 비 + 천둥

4=심한 눈/비, 천둥/번개

temp: 온도(섭씨)

atemp: 체감온도(섭씨)

humidity: 상대습도

windspeed: 풍속

casual: 사전에 등록되지 않는 사용자가 대여한 횟수

registered: 사전에 등록된 사용자가 대여한 횟수

count: 대여 횟수 ⬅ Target

회귀 실습 – 자전거 대여 수요 예측

✓ 데이터 클렌징 및 가공

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

bike_df = pd.read_csv('./bike_train.csv')
print(bike_df.shape)
bike_df.head(3)
```

(10886, 12)

```
bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
datetime      10886 non-null object
season        10886 non-null int64
holiday       10886 non-null int64
workingday    10886 non-null int64
weather       10886 non-null int64
temp          10886 non-null float64
atemp         10886 non-null float64
humidity      10886 non-null int64
windspeed     10886 non-null float64
casual        10886 non-null int64
registered    10886 non-null int64
count         10886 non-null int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

-> datetime 타입으로 변환!

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32

회귀 실습 – 자전거 대여 수요 예측

문자열을 datetime 타입으로 변경

```
# 문자열을 datetime 타입으로 변경.
bike_df['datetime'] = bike_df.datetime.apply(pd.to_datetime)

# datetime 타입에서 년, 월, 일, 시간 추출
bike_df['year'] = bike_df.datetime.apply(lambda x : x.year)
bike_df['month'] = bike_df.datetime.apply(lambda x : x.month)
bike_df['day'] = bike_df.datetime.apply(lambda x : x.day)
bike_df['hour'] = bike_df.datetime.apply(lambda x: x.hour)
bike_df.head(3)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	day	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011	1	1	2

casual + registered = count

필요하지 않은 칼럼 삭제

```
drop_columns = ['datetime','casual','registered']
bike_df.drop(drop_columns, axis=1,inplace=True)
```

회귀 실습 – 자전거 대여 수요 예측

✓ RMSLE, MSE, RMSE 수행하는 성능 평가 함수

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# log 값 변환 시 NaN등의 이슈로 log() 가 아닌 log1p() 를 이용하여 RMSLE 계산
def rmsle(y, pred):
    log_y = np.log1p(y)
    log_pred = np.log1p(pred)
    squared_error = (log_y - log_pred) ** 2
    rmsle = np.sqrt(np.mean(squared_error))
    return rmsle

# 사이킷런의 mean_square_error() 를 이용하여 RMSE 계산
def rmse(y,pred):
    return np.sqrt(mean_squared_error(y,pred))

# MSE, RMSE, RMSLE 를 모두 계산
def evaluate_regr(y,pred):
    rmsle_val = rmsle(y,pred)
    rmse_val = rmse(y,pred)
    # MAE 는 scikit learn의 mean_absolute_error() 로 계산
    mae_val = mean_absolute_error(y,pred)
    print('RMSLE: {0:.3f}, RMSE: {1:.3F}, MAE: {2:.3F}'.format(rmsle_val, rmse_val, mae_val))
```

RMSLE(Root Mean Square Log Error)

:오류 값의 로그에 대한 RMSE

log() 함수

log1p() 함수: 1 + log()

expm1() 함수: 원래의 스케일로 복원

회귀 실습 – 자전거 대여 수요 예측

✓ 로그 변환, 피처 인코딩과 모델 학습/예측/평가 -> 선형 회귀 vs 트리 기반 회귀

LinearRegression 회귀 예측 **[선형 회귀]**

```
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.linear_model import LinearRegression , Ridge , Lasso

y_target = bike_df['count']
X_features = bike_df.drop(['count'],axis=1,inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_features, y_target,
test_size=0.3, random_state=0)

lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

evaluate_regr(y_test ,pred)
```

```
RMSLE: 1.165, RMSE: 140.900, MAE: 105.924
```

실제 값과 예측 값의 차이 확인 (상위 5개)

```
def get_top_error_data(y_test, pred, n_tops = 5):
    # DataFrame에 컬럼들로 실제 대여횟수(count)와 예측 값을 서로 비교 할 수 있도록 생성.
    result_df = pd.DataFrame(y_test.values, columns=['real_count'])
    result_df['predicted_count']= np.round(pred)
    result_df['diff'] = np.abs(result_df['real_count'] - result_df['predicted_count'])
    # 예측값과 실제값이 가장 큰 데이터 순으로 출력.
    print(result_df.sort_values('diff', ascending=False)[:n_tops])

get_top_error_data(y_test,pred,n_tops=5)
```

	real_count	predicted_count	diff
1618	890	322.0	568.0
3151	798	241.0	557.0
966	884	327.0	557.0
412	745	194.0	551.0
2817	856	310.0	546.0

예측 오류 큼!

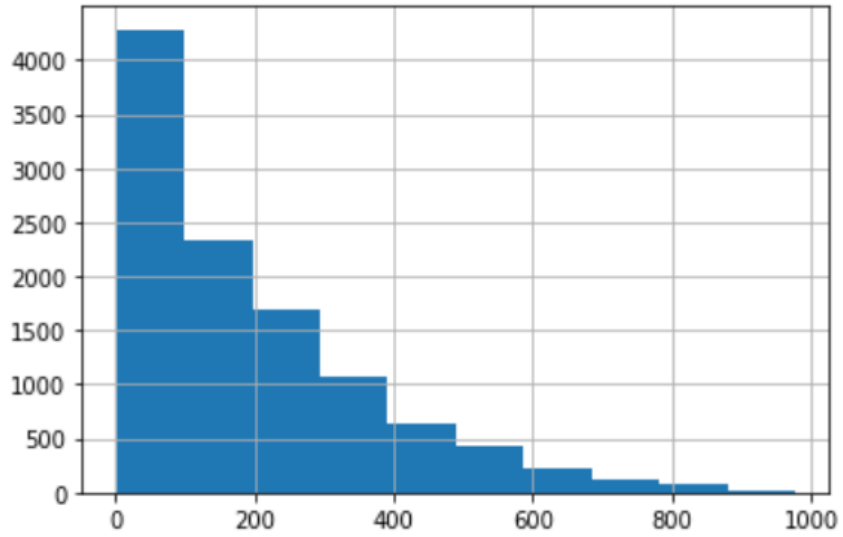
모델 적용 전,
#1 결과값이 정규 분포로 되어 있는지 확인
#2 카테고리형 피처는 원-핫 인코딩 적용

회귀 실습 – 자전거 대여 수요 예측

#1 Target 값인 count 칼럼이 정규 분포를 이루는지 확인

```
y_target.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x223066ff0b8>
```

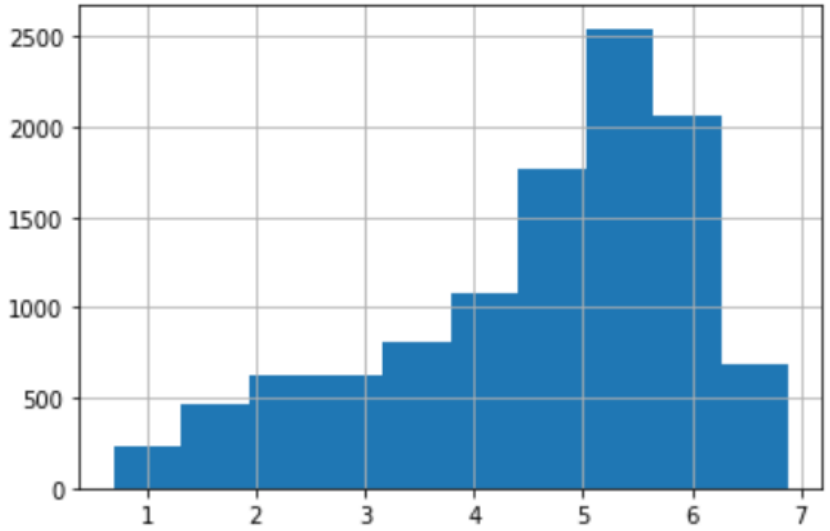


0~200 사이에 왜곡

Log1p()를 적용해 왜곡된 값을 정규 분포 형태로 바꾸기

```
y_log_transform = np.log1p(y_target)
y_log_transform.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2230680bd30>
```



회귀 실습 – 자전거 대여 수요 예측

다시 학습한 후 평가 수행

```
# 타겟 컬럼인 count 값을 log1p 로 Log 변환
y_target_log = np.log1p(y_target)

# 로그 변환된 y_target_log를 반영하여 학습/테스트 데이터 셋 분할
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target_log, test_size=0.3, random_state=0)
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

# 테스트 데이터 셋의 Target 값은 Log 변환되었으므로 다시 expm1를 이용하여 원래 scale로 변환
y_test_exp = np.expm1(y_test)

# 예측 값 역시 Log 변환된 타겟 기반으로 학습되어 예측되었으므로 다시 exmp1으로 scale변환
pred_exp = np.expm1(pred)

evaluate_regr(y_test_exp ,pred_exp)
```

```
RMSLE: 1.017, RMSE: 162.594, MAE: 109.286
```

RMSLE 오류 ↓ but RMSE는 ↑　　그 이유는?

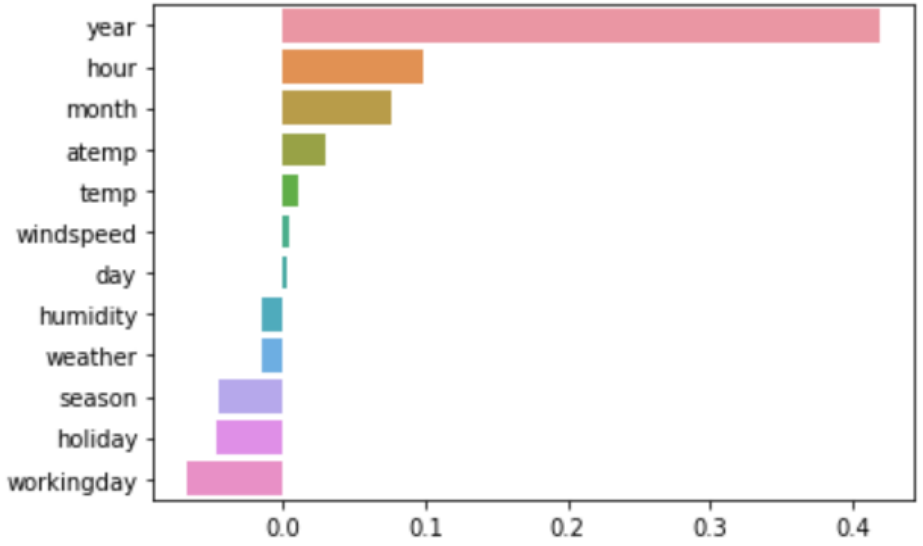
```
RMSLE: 1.165, RMSE: 140.900, MAE: 105.924
```

회귀 실습 – 자전거 대여 수요 예측

각 피처의 회귀 계수 값 시각화

```
coef = pd.Series(lr_reg.coef_, index=X_features.columns)
coef_sort = coef.sort_values(ascending=False)
sns.barplot(x=coef_sort.values, y=coef_sort.index)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2236e248438>
```



Year의 값: 2011, 2012

- 카테고리형 피처지만, 숫자형 값

-> 인코딩 필요!

숫자형 카테고리 값을 선형 회귀에 사용할 경우

회귀 계수를 연산할 때 숫자형 값에 크게 영향을 받음.

따라서 선형 회귀에서는 이러한 피처 인코딩에 원-핫 인코딩을 적용해 변환해야 함.

회귀 실습 – 자전거 대여 수요 예측

#2 원-핫 인코딩한 후 다시 예측 성능 확인

```
# 'year', month', 'day', hour'등의 피처들을 One Hot Encoding
X_features_ohe = pd.get_dummies(X_features, columns=['year', 'month','day', 'hour', 'holiday',
                                              'workingday','season','weather'])
```

```
# 원-핫 인코딩이 적용된 feature 데이터 세트 기반으로 학습/예측 데이터 분할.
X_train, X_test, y_train, y_test = train_test_split(X_features_ohe, y_target_log,
                                                    test_size=0.3, random_state=0)

# 모델과 학습/테스트 데이터 셋을 입력하면 성능 평가 수치를 반환
def get_model_predict(model, X_train, X_test, y_train, y_test, is_expm1=False):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    if is_expm1 :
        y_test = np.expm1(y_test)
        pred = np.expm1(pred)
    print('###',model.__class__.__name__,'###')
    evaluate_regr(y_test, pred)
# end of function get_model_predict

# model 별로 평가 수행
lr_reg = LinearRegression()
ridge_reg = Ridge(alpha=10)
lasso_reg = Lasso(alpha=0.01)

for model in [lr_reg, ridge_reg, lasso_reg]:
    get_model_predict(model,X_train, X_test, y_train, y_test,is_expm1=True)
```

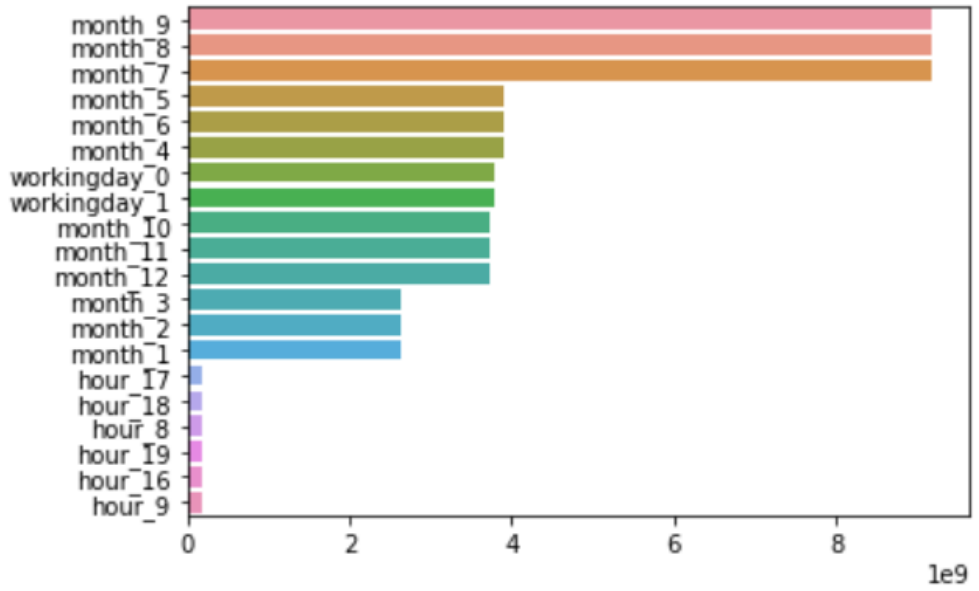
```
### LinearRegression ###
RMSLE: 0.590, RMSE: 97.688, MAE: 63.382
### Ridge ###
RMSLE: 0.590, RMSE: 98.529, MAE: 63.893
### Lasso ###
RMSLE: 0.635, RMSE: 113.219, MAE: 72.803
```

회귀 실습 – 자전거 대여 수요 예측

회귀 계수 상위 25개 피처 시각화

```
coef = pd.Series(lr_reg.coef_ , index=X_features_ohe.columns)
coef_sort = coef.sort_values(ascending=False)[:20]
sns.barplot(x=coef_sort.values , y=coef_sort.index)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22306558ac8>
```



월, 주말/주중, 시간대 등

상식선에서 자전거를 타는데 필요한 피처의 회귀계수가 높아짐

-> 선형 회귀 시 피처를 어떻게 인코딩하는가가 성능에 큰 영향

회귀 실습 – 자전거 대여 수요 예측

회귀 트리를 이용해 회귀 예측 수행 [회귀 트리]

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# 랜덤 포레스트, GBM, XGBoost, LightGBM model 별로 평가 수행
rf_reg = RandomForestRegressor(n_estimators=500)
gbm_reg = GradientBoostingRegressor(n_estimators=500)
xgb_reg = XGBRegressor(n_estimators=500)
lgbm_reg = LGBMRegressor(n_estimators=500)

for model in [rf_reg, gbm_reg, xgb_reg, lgbm_reg]:
    # XGBoost의 경우 DataFrame이 입력 될 경우 버전에 따라 오류 발생 가능. ndarray로 변환.
    get_model_predict(model,X_train.values, X_test.values, y_train.values, y_test.values,is_expm1=True)
```

```
### RandomForestRegressor ###
RMSLE: 0.354, RMSE: 50.099, MAE: 31.050
### GradientBoostingRegressor ###
RMSLE: 0.330, RMSE: 53.355, MAE: 32.749
### XGBRegressor ###
RMSLE: 0.345, RMSE: 58.245, MAE: 35.768
### LGBMRegressor ###
RMSLE: 0.319, RMSE: 47.215, MAE: 29.029
```

선형 회귀 vs 회귀 트리

선형 회귀 모델보다 예측 성능 개선

But 회귀 트리가 선형 회귀보다 나은 성능을 가진다는 의미 X

데이터 세트의 유형에 따라 다름

5. 회귀 실습

캐글 주택 가격: 고급 회귀 기법

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

캐글 주택 가격: 고급 회귀 기법(House Prices: Advanced Regression Techniques) 데이터 세트
https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

미국 아이오와 주의 에임스(Ames) 지방의 주택 가격 정보

✓ 데이터 사전 처리(Preprocessing)

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

house_df_org = pd.read_csv('house_price.csv')
house_df = house_df_org.copy()
house_df.head(3)
```

Target

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500

3 rows × 81 columns

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

데이터 세트의 전체 크기, 칼럼의 타입, Null이 있는 칼럼과 건수 내림차순으로 출력

```
print('데이터 세트의 Shape:', house_df.shape)
print('₩n전체 feature 들의 type ₩n',house_df.dtypes.value_counts())
isnull_series = house_df.isnull().sum()
print('₩nNull 컬럼과 그 건수:₩n ', isnull_series[isnull_series > 0].sort_values(ascending=False))
```

```
데이터 세트의 Shape: (1460, 81)

전체 feature 들의 type
 object     43
int64       35
float64      3
dtype: int64
```

Null 값이 너무 많기 때문에 삭제

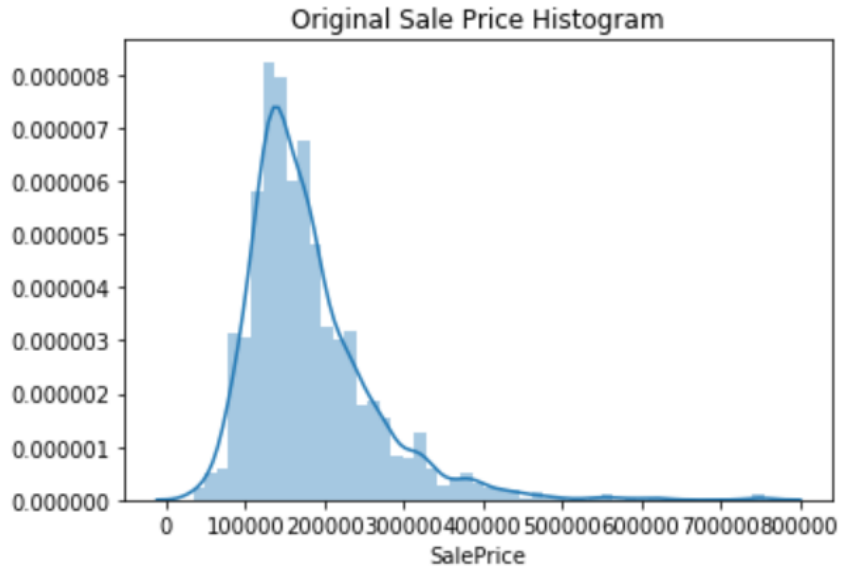
```
Null 컬럼과 그 건수:
  PoolQC          1453
MiscFeature       1406
Alley             1369
Fence             1179
FireplaceQu        690
LotFrontage        259
GarageYrBlt         81
GarageType          81
GarageFinish        81
GarageQual          81
GarageCond          81
BsmtFinType2        38
BsmtExposure        38
BsmtFinType1        37
BsmtCond            37
BsmtQual            37
MasVnrArea           8
MasVnrType           8
Electrical           1
dtype: int64
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

타깃 값의 분포도가 정규 분포인지 확인

```
plt.title('Original Sale Price Histogram')
sns.distplot(house_df['SalePrice'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2279325d940>
```



로그 변환(Log Transformation) 적용

```
plt.title('Log Transformed Sale Price Histogram')
log_SalePrice = np.log1p(house_df['SalePrice'])
sns.distplot(log_SalePrice)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x222e73d4e48>
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

Null 값이 많은 피처 처리

```
# SalePrice 로그 변환
original_SalePrice = house_df['SalePrice']
house_df['SalePrice'] = np.log1p(house_df['SalePrice'])

# Null 이 너무 많은 컬럼들과 불필요한 컬럼 삭제
house_df.drop(['Id','PoolQC' , 'MiscFeature', 'Alley', 'Fence','FireplaceQu'], axis=1 , inplace=True)
# Drop 하지 않는 숫자형 Null컬럼들은 평균값으로 대체
house_df.fillna(house_df.mean(),inplace=True)

# Null 값이 있는 피처명과 타입을 추출
null_column_count = house_df.isnull().sum()[house_df.isnull().sum() > 0]
print('## Null 피처의 Type :₩n', house_df.dtypes[null_column_count.index])
```

```
## Null 피처의 Type :
 MasVnrType      object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinType2     object
Electrical       object
GarageType       object
GarageFinish     object
GarageQual       object
GarageCond       object
dtype: object
```

문자형 피처 원-핫 인코딩으로 변환

```
print('get_dummies() 수행 전 데이터 Shape:', house_df.shape)
house_df_ohe = pd.get_dummies(house_df)
print('get_dummies() 수행 후 데이터 Shape:', house_df_ohe.shape)

null_column_count = house_df_ohe.isnull().sum()[house_df_ohe.isnull().sum() > 0]
print('## Null 피처의 Type :₩n', house_df_ohe.dtypes[null_column_count.index])
```

```
get_dummies() 수행 전 데이터 Shape: (1460, 75)
get_dummies() 수행 후 데이터 Shape: (1460, 271)
## Null 피처의 Type :
 Series([], dtype: object)
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

✓ 선형 회귀 모델 학습/예측/평가

RMSE를 계산하는 함수 생성

```
def get_rmse(model):
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test , pred)
    rmse = np.sqrt(mse)
    print('{0} 로그 변환된 RMSE: {1}'.format(model.__class__.__name__,np.round(rmse, 3)))
    return rmse

def get_rmses(models):
    rmses = [ ]
    for model in models:
        rmse = get_rmse(model)
        rmses.append(rmse)
    return rmses
```

get_rmse(model)
단일 모델의 RMSE 값 반환

get_rmses(models)
여러 모델의 RMSE 값 반환

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

선형 회귀 모델 학습/예측/평가

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

y_target = house_df_ohe['SalePrice']
X_features = house_df_ohe.drop('SalePrice',axis=1, inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=156)

# LinearRegression, Ridge, Lasso 학습, 예측, 평가
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)

ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)

lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)

models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

```
LinearRegression 로그 변환된 RMSE: 0.132
Ridge 로그 변환된 RMSE: 0.128
Lasso 로그 변환된 RMSE: 0.176
[0.131895765791542, 0.12750846334053026, 0.17628250556471395]
```

문제: 라쏘 회귀 성능 떨어짐

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

피처별 회귀 계수 시각화

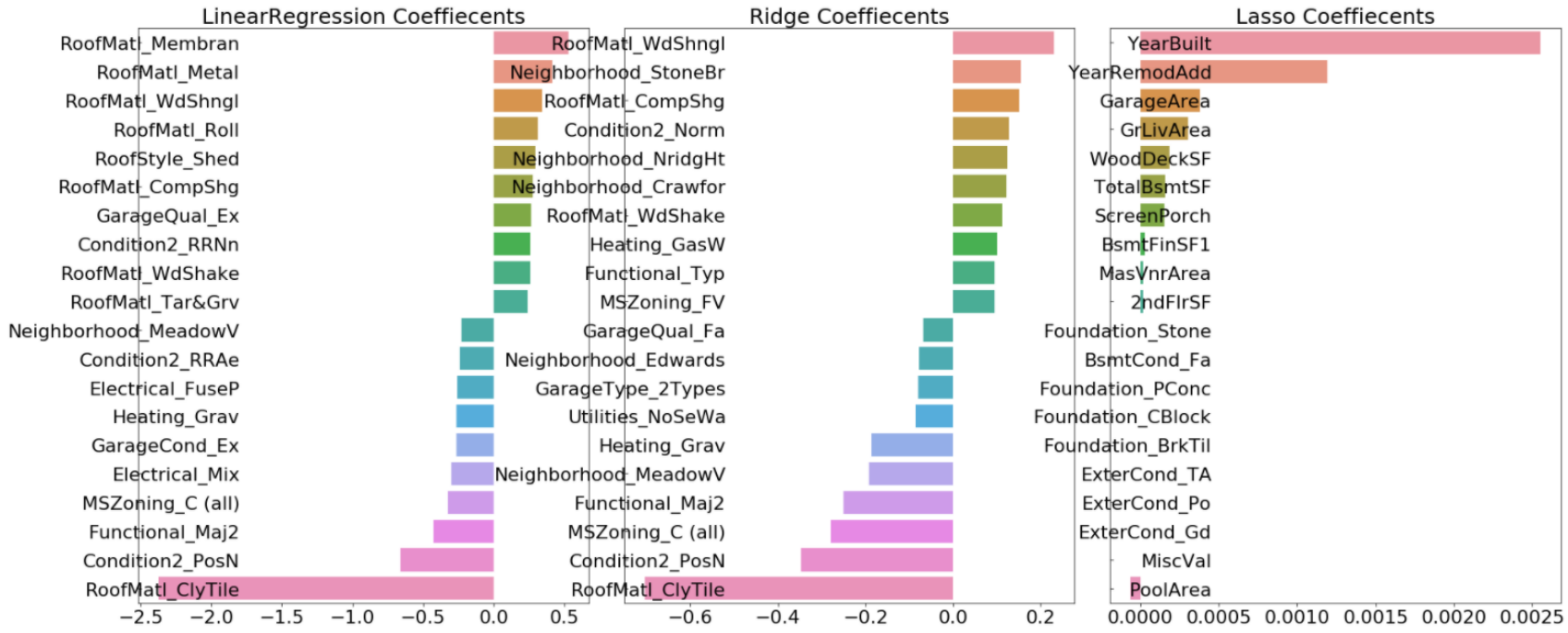
```
def get_top_bottom_coef(model):
    # coef_ 속성을 기반으로 Series 객체를 생성. index는 컬럼명.
    coef = pd.Series(model.coef_, index=X_features.columns)

    # + 상위 10개 , - 하위 10개 coefficient 추출하여 반환.
    coef_high = coef.sort_values(ascending=False).head(10)
    coef_low = coef.sort_values(ascending=False).tail(10)
    return coef_high, coef_low
```

회귀 계수 값의 상위 10개, 하위 10개의 피처명과
그 회귀 계수를 가지는 시리즈 객체 반환 함수

```
def visualize_coefficient(models):
    # 3개 회귀 모델의 시각화를 위해 3개의 컬럼을 가지는 subplot 생성
    fig, axs = plt.subplots(figsize=(24,10),nrows=1, ncols=3)
    fig.tight_layout()
    # 입력인자로 받은 list객체인 models에서 차례로 model을 추출하여 회귀 계수 시각화.
    for i_num, model in enumerate(models):
        # 상위 10개, 하위 10개 회귀 계수를 구하고, 이를 판다스 concat으로 결합.
        coef_high, coef_low = get_top_bottom_coef(model)
        coef_concat = pd.concat( [coef_high , coef_low] )
        # 순차적으로 ax subplot에 barchar로 표현. 한 화면에 표현하기 위해 tick label 위치와 font 크기 조정.
        axs[i_num].set_title(model.__class__.__name__+' Coeffiecents', size=25)
        axs[i_num].tick_params(axis="y",direction="in", pad=-120)
        for label in (axs[i_num].get_xticklabels() + axs[i_num].get_yticklabels()):
            label.set_fontsize(22)
        sns.barplot(x=coef_concat.values, y=coef_concat.index , ax=axs[i_num])

# 앞 예제에서 학습한 lr_reg, ridge_reg, lasso_reg 모델의 회귀 계수 시각화.
models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

결과는 다음 페이지에…

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법



라쏘만 다른 회귀 계수의 형태

-> 학습 데이터의 데이터 분할 문제?

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

#1 전체 데이터세트를 5개의 **교차 검증 폴드 세트로 분할**해 평균 RMSE 측정

```
from sklearn.model_selection import cross_val_score

def get_avg_rmse_cv(models):
    for model in models:
        # 분할하지 않고 전체 데이터로 cross_val_score( ) 수행. 모델별 CV RMSE값과 평균 RMSE 출력
        rmse_list = np.sqrt(-cross_val_score(model, X_features, y_target,
                                             scoring="neg_mean_squared_error", cv = 5))
        rmse_avg = np.mean(rmse_list)
        print('₩n{0} CV RMSE 값 리스트: {1}'.format( model.__class__.__name__, np.round(rmse_list, 3)))
        print('{0} CV 평균 RMSE 값: {1}'.format( model.__class__.__name__, np.round(rmse_avg, 3)))

# 앞 예제에서 학습한 lr_reg, ridge_reg, lasso_reg 모델의 CV RMSE값 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_avg_rmse_cv(models)
```

```
LinearRegression CV RMSE 값 리스트: [0.135 0.165 0.168 0.111 0.198]
LinearRegression CV 평균 RMSE 값: 0.155

Ridge CV RMSE 값 리스트: [0.117 0.154 0.142 0.117 0.189]
Ridge CV 평균 RMSE 값: 0.144

Lasso CV RMSE 값 리스트: [0.161 0.204 0.177 0.181 0.265]
Lasso CV 평균 RMSE 값: 0.198
```

여전히 라쏘의 성능 떨어짐

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

#2 모델별로 최적화 하이퍼 파라미터 작업

```
from sklearn.model_selection import GridSearchCV

def print_best_params(model, params):
    grid_model = GridSearchCV(model, param_grid=params,
                              scoring='neg_mean_squared_error', cv=5)
    grid_model.fit(X_features, y_target)
    rmse = np.sqrt(-1* grid_model.best_score_)
    print('{0} 5 CV 시 최적 평균 RMSE 값: {1}, 최적 alpha:{2}'.format(model.__class__.__name__,
                                    np.round(rmse, 4), grid_model.best_params_))
    return grid_model.best_estimator_

ridge_params = { 'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20] }
lasso_params = { 'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1,5, 10] }
best_rige = print_best_params(ridge_reg, ridge_params)
best_lasso = print_best_params(lasso_reg, lasso_params)
```

모델과 하이퍼 파라미터 딕셔너리 객체를 받아 최적화 작업의 결과 표시하는 함수

```
Ridge 5 CV 시 최적 평균 RMSE 값: 0.1418, 최적 alpha:{'alpha': 12}
Lasso 5 CV 시 최적 평균 RMSE 값: 0.142, 최적 alpha:{'alpha': 0.001}
```

라쏘 성능 향상

alpha 하이퍼 파라미터 변화시키면서 릿지와 라쏘의 결과 확인
-> 최적의 alpha 값 추출

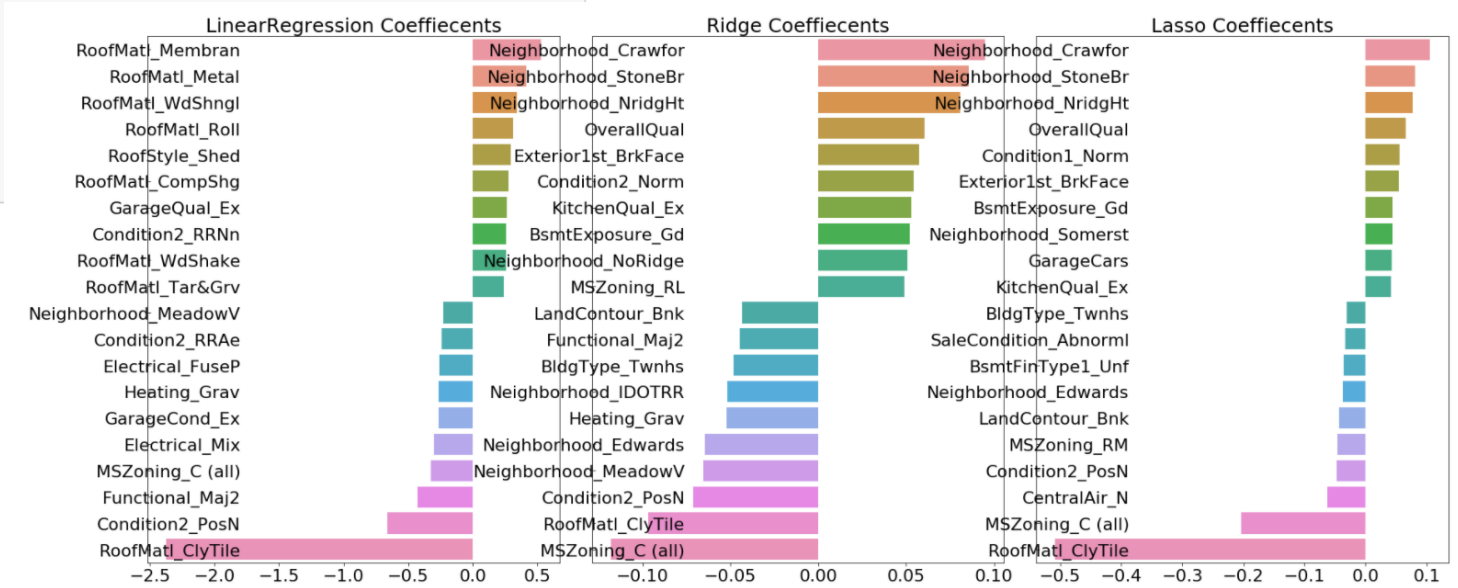
회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

alpha 값 최적화 후 예측 성능 확인

```
# 앞의 최적화 alpha값으로 학습데이터로 학습, 테스트 데이터로 예측 및 평가 수행.
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge(alpha=12)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)

# 모든 모델의 RMSE 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

# 모든 모델의 회귀 계수 시각화
models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

```
LinearRegression 로그 변환된 RMSE: 0.132
Ridge 로그 변환된 RMSE: 0.124
Lasso 로그 변환된 RMSE: 0.12
```

라쏘의 예측 성능 좋아짐



회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

추가 가공 #1 피처 데이터 세트의 분포도

STEP1) 숫자형 피처 데이터 세트의 왜곡된 정도 추출

```
from scipy.stats import skew

# object가 아닌 숫자형 피쳐의 컬럼 index 객체 추출.
features_index = house_df.dtypes[house_df.dtypes != 'object'].index
# house_df에 컬럼 index를 [ ]로 입력하면 해당하는 컬럼 데이터 셋 반환. apply lambda로 skew( )호출
skew_features = house_df[features_index].apply(lambda x : skew(x))
# skew 정도가 1 이상인 컬럼들만 추출.
skew_features_top = skew_features[skew_features > 1]
print(skew_features_top.sort_values(ascending=False))
```

skew() 함수의 반환 값이 1이상 -> 왜곡 정도 높음

```
MiscVal          24.451640
PoolArea         14.813135
LotArea          12.195142
3SsnPorch        10.293752
LowQualFinSF      9.002080
KitchenAbvGr      4.483784
BsmtFinSF2        4.250888
ScreenPorch       4.117977
BsmtHalfBath      4.099186
EnclosedPorch     3.086696
MasVnrArea        2.673661
LotFrontage       2.382499
OpenPorchSF       2.361912
BsmtFinSF1        1.683771
WoodDeckSF        1.539792
TotalBsmtSF       1.522688
MSSubClass        1.406210
1stFlrSF          1.375342
GrLivArea         1.365156
dtype: float64
```

주의! Skew()를 적용하는 숫자형 피처에서
원-핫 인코딩된 카테고리 숫자형 피처는 제외

STEP2) 왜곡 정도가 높은 피처 로그 변환

```
house_df[skew_features_top.index] = np.log1p(house_df[skew_features_top.index])
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

다시 원-핫 인코딩 적용 및 피처/타겟 데이터 세트 생성, 최적 alpha 값과 RMSE 출력

```
# Skew가 높은 피처들을 로그 변환 했으므로 다시 원-핫 인코딩 적용 및 피처/타겟 데이터 셋 생성,
house_df_ohe = pd.get_dummies(house_df)
y_target = house_df_ohe['SalePrice']
X_features = house_df_ohe.drop('SalePrice',axis=1, inplace=False)
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=156)

# 피처들을 로그 변환 후 다시 최적 하이퍼 파라미터와 RMSE 출력
ridge_params = { 'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20] }
lasso_params = { 'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1,5, 10] }
best_ridge = print_best_params(ridge_reg, ridge_params)
best_lasso = print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 CV 시 최적 평균 RMSE 값: 0.1275, 최적 alpha:{'alpha': 10}
Lasso 5 CV 시 최적 평균 RMSE 값: 0.1252, 최적 alpha:{'alpha': 0.001}
```

5 폴드 교차 검증의 평균 RMSE값 향상

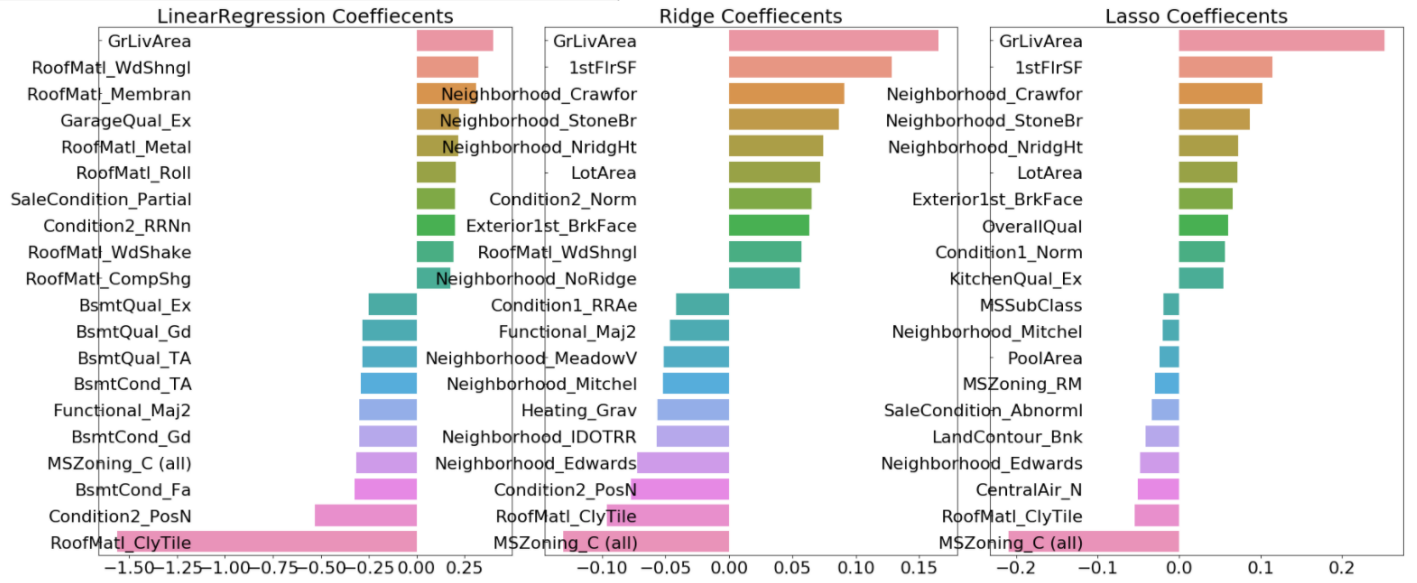
회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

모델의 학습/예측/평가 및 모델별 회귀 계수 시각화

```
# 앞의 최적화 alpha값으로 학습데이터로 학습, 테스트 데이터로 예측 및 평가 수행.
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge(alpha=10)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)

# 모든 모델의 RMSE 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

# 모든 모델의 회귀 계수 시각화
models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

```
LinearRegression 로그 변환된 RMSE: 0.128
Ridge 로그 변환된 RMSE: 0.122
Lasso 로그 변환된 RMSE: 0.119
```

회귀 계수가 가장 높은 피처
-> GrLivArea(주거 공간 크기)

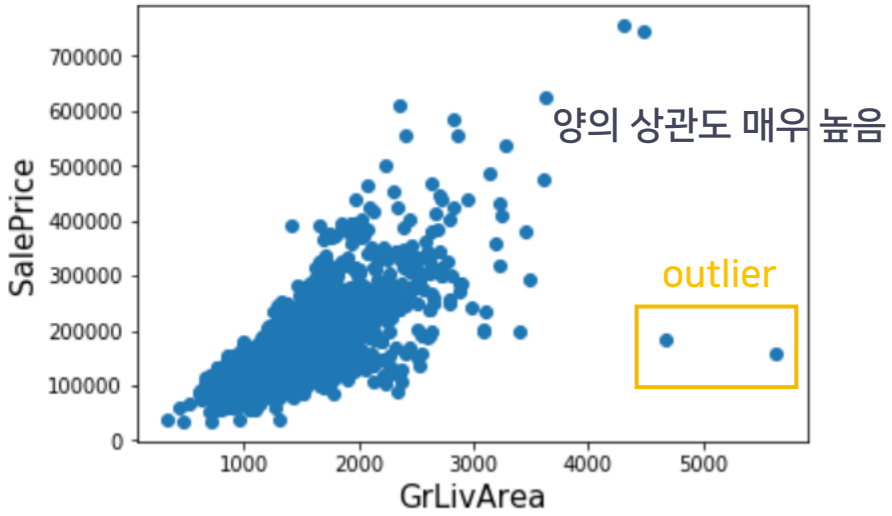


회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

추가 가공 #2 이상치(Outlier) 데이터 처리

STEP1) 회귀 계수가 가장 높은 중요 피처 GrLivArea의 데이터 분포 확인

```
plt.scatter(x = house_df_org['GrLivArea'], y = house_df_org['SalePrice'])
plt.ylabel('SalePrice', fontsize=15)
plt.xlabel('GrLivArea', fontsize=15)
plt.show()
```



STEP2) 이상치 데이터를 찾아 삭제

```
# GrLivArea와 SalePrice 모두 로그 변환되었으므로 이를 반영한 조건 생성.
cond1 = house_df_ohe['GrLivArea'] > np.log1p(4000)
cond2 = house_df_ohe['SalePrice'] < np.log1p(500000)
outlier_index = house_df_ohe[cond1 & cond2].index

print('아웃라이어 레코드 index :', outlier_index.values)
print('아웃라이어 삭제 전 house_df_ohe shape:', house_df_ohe.shape)
# DataFrame의 index를 이용하여 아웃라이어 레코드 삭제.
house_df_ohe.drop(outlier_index , axis=0, inplace=True)
print('아웃라이어 삭제 후 house_df_ohe shape:', house_df_ohe.shape)
```

```
아웃라이어 레코드 index : [ 523 1298]
아웃라이어 삭제 전 house_df_ohe shape: (1460, 271)
아웃라이어 삭제 후 house_df_ohe shape: (1458, 271)
```

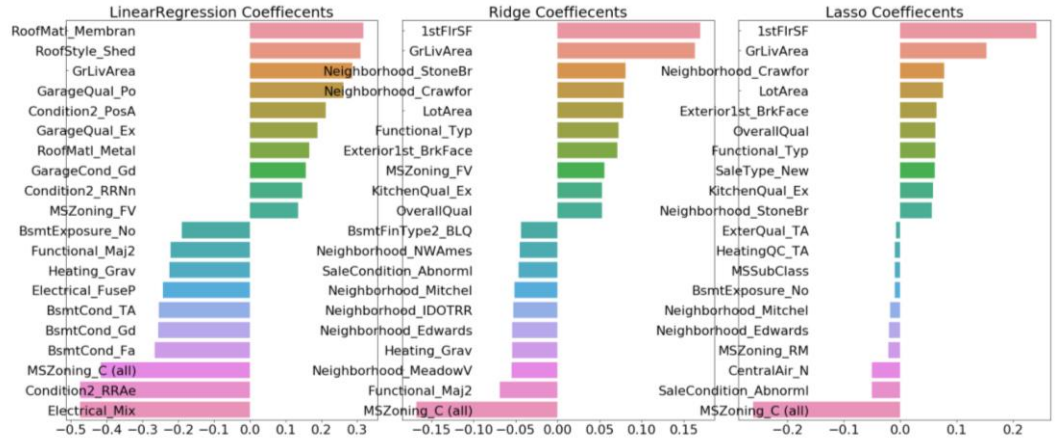
회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

다시 피처/타겟 데이터 세트 생성, 최적 alpha 값과 RMSE 출력

```
y_target = house_df_ohe['SalePrice']
X_features = house_df_ohe.drop('SalePrice',axis=1, inplace=False)
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=156)

ridge_params = { 'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20] }
lasso_params = { 'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1,5, 10] }
best_ridge = print_best_params(ridge_reg, ridge_params)
best_lasso = print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 CV 시 최적 평균 RMSE 값: 0.1125, 최적 alpha:{'alpha': 8}
Lasso 5 CV 시 최적 평균 RMSE 값: 0.1122, 최적 alpha:{'alpha': 0.001}
```

예측 수치 크게 향상



모델의 학습/예측/평가 및 모델별 회귀 계수 시각화

```
# 앞의 최적화 alpha값으로 학습데이터로 학습, 테스트 데이터로 예측 및 평가 수행.
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge(alpha=8)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)

# 모든 모델의 RMSE 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

# 모든 모델의 회귀 계수 시각화
models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

```
LinearRegression 로그 변환된 RMSE: 0.129
Ridge 로그 변환된 RMSE: 0.103
Lasso 로그 변환된 RMSE: 0.1
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

✓ 회귀 트리 모델 학습/예측/평가

```
from xgboost import XGBRegressor

xgb_params = {'n_estimators':[1000]}
xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05,
                       colsample_bytree=0.5, subsample=0.8)
best_xgb = print_best_params(xgb_reg, xgb_params)
```

```
XGBRegressor 5 CV 시 최적 평균 RMSE 값: 0.115, 최적 alpha:{'n_estimators': 1000}
```

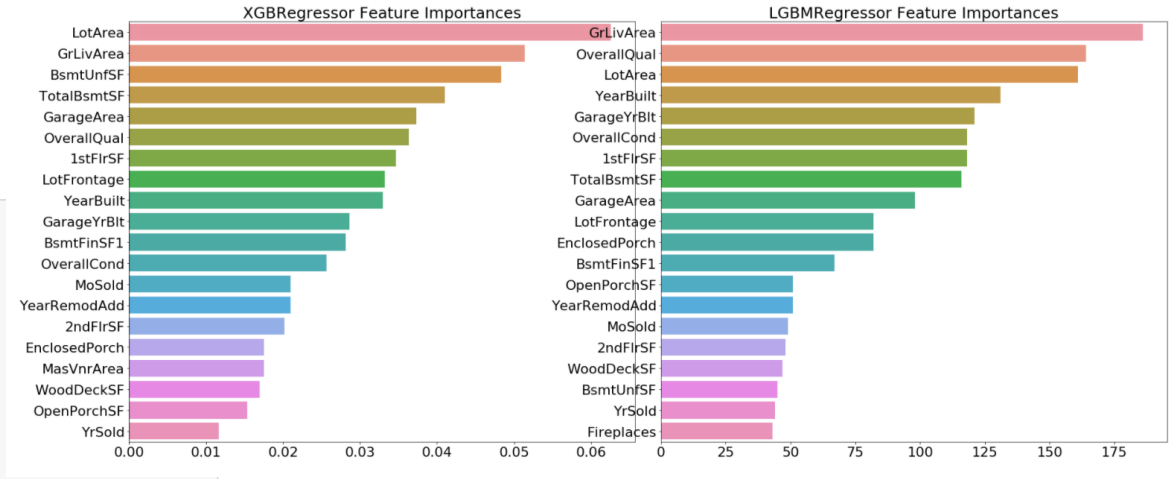
```
from lightgbm import LGBMRegressor

lgbm_params = {'n_estimators':[1000]}
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
best_lgbm = print_best_params(lgbm_reg, lgbm_params)
```

```
LGBMRegressor 5 CV 시 최적 평균 RMSE 값: 0.1161, 최적 alpha:{'n_estimators': 1000}
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

회귀 트리 모델의 피처 중요도 시각화

```
# 모델의 중요도 상위 20개의 피처명과 그때의 중요도값을 Series로 반환.
def get_top_features(model):
    ftr_importances_values = model.feature_importances_
    ftr_importances = pd.Series(ftr_importances_values, index=X_features.columns  )
    ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
    return ftr_top20

def visualize_ftr_importances(models):
    # 2개 회귀 모델의 시각화를 위해 2개의 컬럼을 가지는 subplot 생성
    fig, axs = plt.subplots(figsize=(24,10),nrows=1, ncols=2)
    fig.tight_layout()
    # 입력인자로 받은 list객체인 models에서 차례로 model을 추출하여 피처 중요도 시각화.
    for i_num, model in enumerate(models):
        # 중요도 상위 20개의 피처명과 그때의 중요도값 추출
        ftr_top20 = get_top_features(model)
        axs[i_num].set_title(model.__class__.__name__+' Feature Importances', size=25)
        #font 크기 조정.
        for label in (axs[i_num].get_xticklabels() + axs[i_num].get_yticklabels()):
            label.set_fontsize(22)
        sns.barplot(x=ftr_top20.values, y=ftr_top20.index , ax=axs[i_num])

# 앞 예제에서 print_best_params( )가 반환한 GridSearchCV로 최적화된 모델의 피처 중요도 시각화
models = [best_xgb, best_lgbm]
visualize_ftr_importances(models)
```



회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

✓ 회귀 모델의 예측 결과 혼합을 통한 최종 예측

릿지 모델과 라쏘 모델의 예측 결과 혼합

```
def get_rmse_pred(preds):
    for key in preds.keys():
        pred_value = preds[key]
        mse = mean_squared_error(y_test , pred_value)
        rmse = np.sqrt(mse)
        print('{0} 모델의 RMSE: {1}'.format(key, rmse))

# 개별 모델의 학습
ridge_reg = Ridge(alpha=8)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)
# 개별 모델 예측
ridge_pred = ridge_reg.predict(X_test)
lasso_pred = lasso_reg.predict(X_test)

# 개별 모델 예측값 혼합으로 최종 예측값 도출
pred = 0.4 * ridge_pred + 0.6 * lasso_pred
preds = {'최종 혼합': pred,
         'Ridge': ridge_pred,
         'Lasso': lasso_pred}
#최종 혼합 모델, 개별모델의 RMSE 값 출력
get_rmse_pred(preds)
```

```
최종 혼합 모델의 RMSE: 0.10007930884470517
Ridge 모델의 RMSE: 0.10345177546603249
Lasso 모델의 RMSE: 0.10024170460890039
```

릿지 모델 예측값 x 0.4
+ 라쏘 모델 예측값 x 0.6

최종 혼합 모델, 개별 모델의
RMSE 값 출력하는 함수

예측 결과 혼합

A 모델 예측값 40% + B 모델 예측값 60%

A 회귀 모델의 예측값 [100, 80, 60]
B 회귀 모델의 예측값 [120, 80, 50]

최종 회귀 예측값
[100*0.4 + 120*0.6, 80*0.4 + 80*0.6, 60*0.4 + 50*0.6] =
[112, 80, 54]

-> 특별한 기준 X

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

XGBoost와 LightGBM의 예측 결과 혼합

```
xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05,
                       colsample_bytree=0.5, subsample=0.8)
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
xgb_reg.fit(X_train, y_train)
lgbm_reg.fit(X_train, y_train)
xgb_pred = xgb_reg.predict(X_test)
lgbm_pred = lgbm_reg.predict(X_test)

pred = 0.5 * xgb_pred + 0.5 * lgbm_pred
preds = {'최종 혼합': pred,
         'XGBM': xgb_pred,
         'LGBM': lgbm_pred}

get_rmse_pred(preds)
```

```
최종 혼합 모델의 RMSE: 0.10011146490791507
XGBM 모델의 RMSE: 0.10356482646891171
LGBM 모델의 RMSE: 0.1015065721553885
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

✓ 스태킹 앙상블 모델을 통한 회귀 예측

최종 메타 모델이 사용할
학습 및 테스트용 데이터 생성 함수

스태킹 모델의 구현 방법

여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해
최종 메타 모델의 학습용/테스트용 피처 데이터 세트 만듦

메타 모델이 사용할 학습/테스트 피처 데이터 세트 추출

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

# 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds ):
    # 지정된 n_folds값으로 KFold 생성.
    kf = KFold(n_splits=n_folds, shuffle=False, random_state=0)
    #추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
    train_fold_pred = np.zeros((X_train_n.shape[0] ,1 ))
    test_pred = np.zeros((X_test_n.shape[0],n_folds))
    print(model.__class__.__name__ , ' model 시작 ')

    for folder_counter , (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        #입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 셋 추출
        print('₩t 폴드 세트: ',folder_counter,' 시작 ')
        X_tr = X_train_n[train_index]
        y_tr = y_train_n[train_index]
        X_te = X_train_n[valid_index]

        #폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
        model.fit(X_tr , y_tr)
        #폴드 세트 내부에서 다시 만들어진 검증 데이터로 기반 모델 예측 후 데이터 저장.
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1,1)
        #입력된 원본 테스트 데이터를 폴드 세트내 학습된 기반 모델에서 예측 후 데이터 저장.
        test_pred[:, folder_counter] = model.predict(X_test_n)

    # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)

    #train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
    return train_fold_pred , test_pred_mean
```

회귀 실습 – 캐글 주택 가격: 고급 회귀 기법

```
# get_stacking_base_datasets( )은 넘파이 ndarray를 인자로 사용하므로 DataFrame을 넘파이로 변환.
X_train_n = X_train.values
X_test_n = X_test.values
y_train_n = y_train.values

# 각 개별 기반(Base)모델이 생성한 학습용/테스트용 데이터 반환.
ridge_train, ridge_test = get_stacking_base_datasets(ridge_reg, X_train_n, y_train_n, X_test_n, 5)
lasso_train, lasso_test = get_stacking_base_datasets(lasso_reg, X_train_n, y_train_n, X_test_n, 5)
xgb_train, xgb_test = get_stacking_base_datasets(xgb_reg, X_train_n, y_train_n, X_test_n, 5)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm_reg, X_train_n, y_train_n, X_test_n, 5)
```

```
Ridge  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
Lasso  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
XGBRegressor  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
LGBMRegressor  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
```

최종 메타 모델에 적용해 예측 및 RMSE 측정

```
# 개별 모델이 반환한 학습 및 테스트용 데이터 세트를 Stacking 형태로 결합.
Stack_final_X_train = np.concatenate((ridge_train, lasso_train,
                                      xgb_train, lgbm_train), axis=1)
Stack_final_X_test = np.concatenate((ridge_test, lasso_test,
                                     xgb_test, lgbm_test), axis=1)

# 최종 메타 모델은 라쏘 모델을 적용.
meta_model_lasso = Lasso(alpha=0.0005)

#기반 모델의 예측값을 기반으로 새롭게 만들어진 학습 및 테스트용 데이터로 예측하고 RMSE 측정.
meta_model_lasso.fit(Stack_final_X_train, y_train)
final = meta_model_lasso.predict(Stack_final_X_test)
mse = mean_squared_error(y_test , final)
rmse = np.sqrt(mse)
print('스태킹 회귀 모델의 최종 RMSE 값은:', rmse)
```

```
스태킹 회귀 모델의 최종 RMSE 값은: 0.09723804106266824
```

스태킹 모델은 회귀에서 특히
효과적으로 사용할 수 있는 모델

감사합니다