5장. 회귀

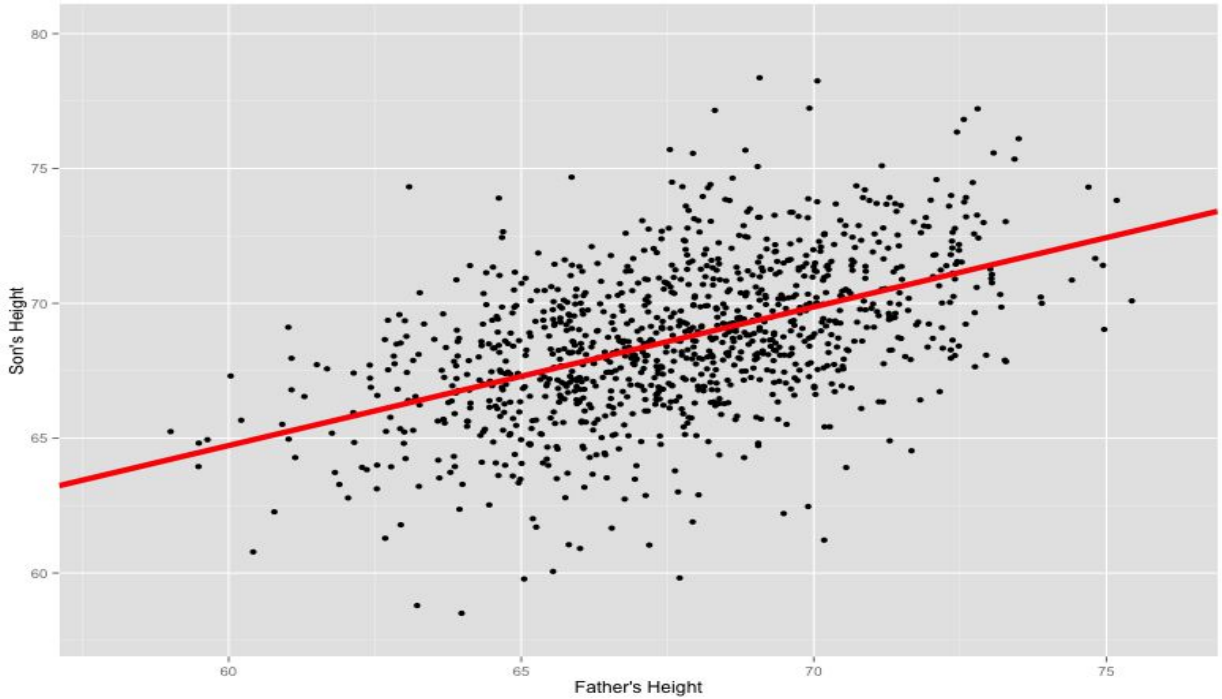
염석준

내용

1. 회귀 소개
2. 경사 하강법
3. 다항 회귀
    a. 과소적합/과대적합
4. 규제 선형 모델
    a. 릿지 회귀
    b. 라쏘 회귀
    c. 엘라스틱넷 회귀
5. 로지스틱 회귀
6. 회귀 트리

1. 회귀란

- 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법



- 회귀에서 가장 중요한 것은 회귀 계수

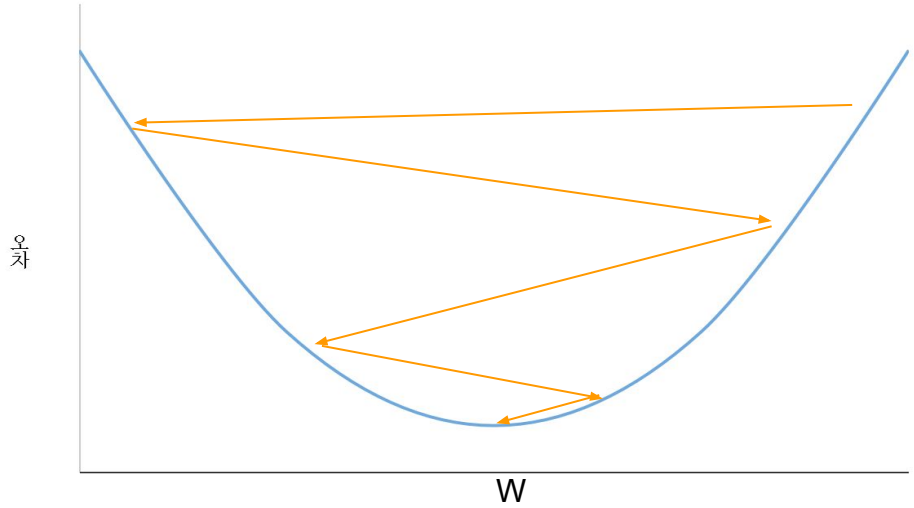
독립변수 개수	회귀 계수의 결합
1개 : 단일 회귀	선형 : 선형 회귀
여러 개 : 다중 회귀	비선형 : 비선형 회귀

- 지도학습의 한 종류, 분류는 예측값이 카테고리와 같은 이산형 클래스, 회귀는 연속형 숫자 값
- RSS = error^2 이며 이 값을 최소화 하는 것이 머신러닝 기반 회귀의 핵심이며, 회귀에서 RSS 는 비용
- 회귀계수로 구성되는 RSS 를 비용 함수 = 손실 함수

2. 경사 하강법

- ‘점진적으로’ 반복적인 계산을 통해 W 파라미터(회귀 변수) 값을 업데이트 하면서 오류 값이 최소가 되는 W 파라미터를 구하는 방식

a. 앞을 보기 힘들 정도로, 안개가 낀 협곡에 있다고 가정
b. 앞이 보이지 않아, 협곡을 내려가는 길을 찾을 수가 없지만, 바로 앞에서의 협곡의 경사는 확인할 수는 있음
c. 경사가 내려가면 해당 방향으로 가고, 경사가 올라가면 내려가는 경사를 찾아 방향을 정함
d. 이렇게 꾸준히 조심히 내려가다 보면, 언젠가는 바닥에 도착



- 핵심은 '어떻게 하면 오류가 작아지는 방향으로 W 값을 보정할 수 있을까'
- 모든 학습데이터에 대해 반복적으로 비용함수 최소화를 위한 값을 업데이트하기 때문에 수행시간이 매우 오래 걸린다는 단점이 있음
- 일부 데이터만을 이용해서 계산하는 확률적 경사 하강법을 이용

- LinearRegression 클래스

```
class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)
```

입력 파라미터	fit_intercept	Boolean값, 디폴트:True, 절편 값을 계산할 것인지 말지를 지정 False 일 경우 절편은 0으로 지정
	normalize	Boolean값, 디폴트:False, fit_intercept 가 False 인 경우 이 파라미터가 무시, True 이면 회귀를 수행하기 전에 입력 데이터 세트를 정규화
속성	coef_	fit() 메소드를 수행했을 때 회귀 계수가 배열 형태로 저장하는 속성
	intercept_	intercept 값

- 회귀 평가 지표

평가 지표	설명
MAE (Mean Absolute Error)	실제 값과 예측값의 차이를 절댓값으로 변환해 평균
MSE (Mean Squared Error)	실제 값과 예측값의 차이를 제곱해 평균
RMSE (Root Mean Squared Error)	MSE 값은 오류의 제곱을 구하므로 실제 오류 평균보다 커지는 특성이 있으므로 MSE 에 루트를 씌운 것
R^2	분산 기반으로 예측 성능을 평가하며 1에 가까울 수록 예측 정확도가 높음

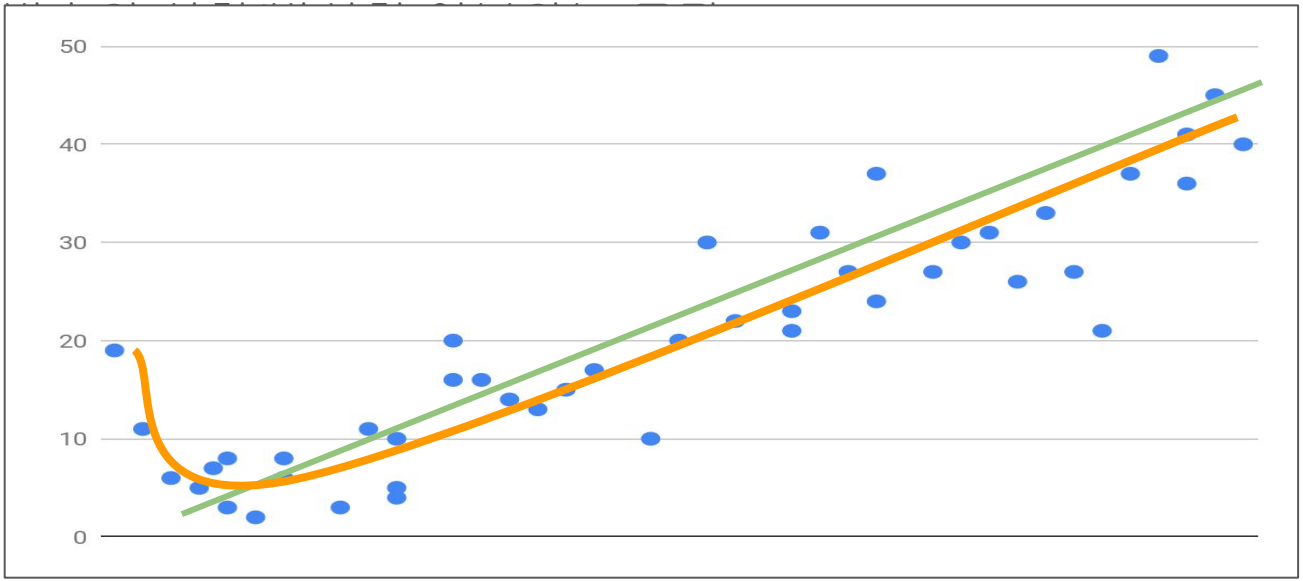
```
from sklearn.model_selection import cross_val_score

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

# cross_val_score( )로 5 Fold 셋으로 MSE 를 구한 뒤 이를 기반으로 다시  RMSE 구함.
neg_mse_scores = cross_val_score(lr, 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)

# cross_val_score(scoring="neg_mean_squared_error")로 반환된 값은 모두 음수
print(' 5 folds 의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 2))
print(' 5 folds 의 개별 RMSE scores : ', np.round(rmse_scores, 2))
print(' 5 folds 의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

```
 5 folds 의 개별 Negative MSE scores:  [-12.46 -26.05 -33.07 -80.76 -33.31]
 5 folds 의 개별 RMSE scores :  [3.53 5.1  5.75 8.99 5.77]
 5 folds 의 평균 RMSE : 5.829
```

- LinearRegression 을 이용한 보스턴 주택 가격 예측

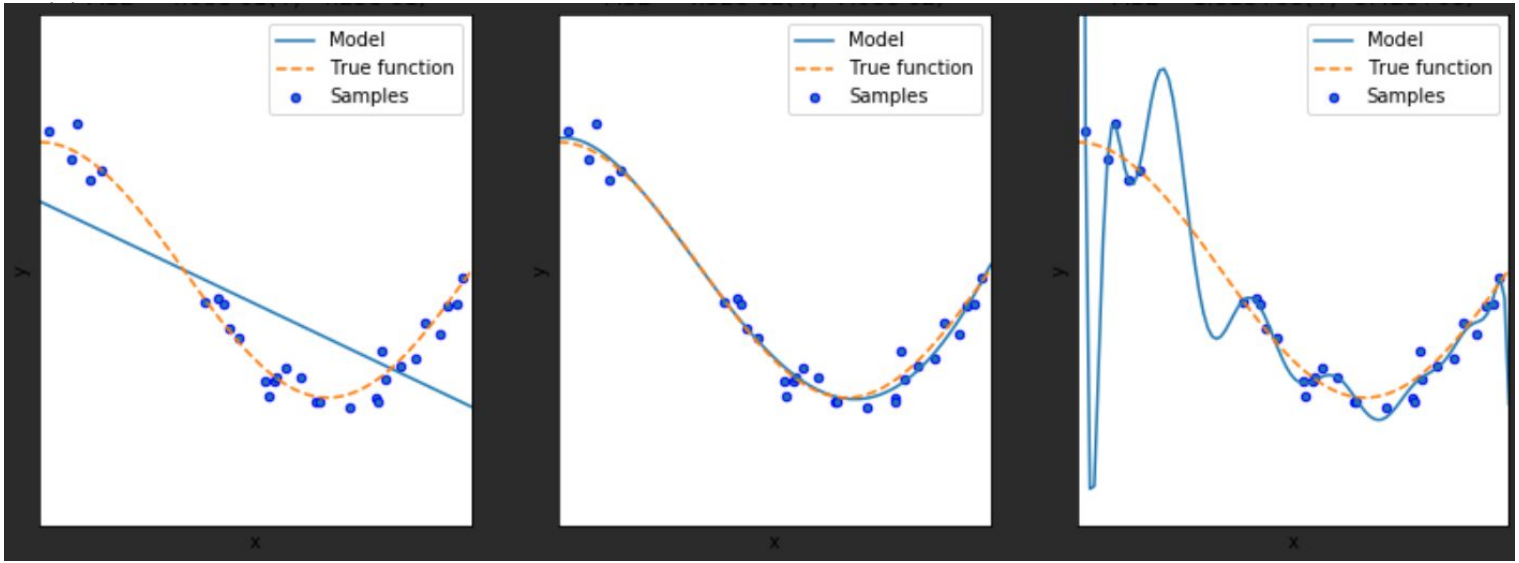
3. 다항 회귀

- 회귀가 독립변수의 단항식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 형태
- 선형/비선형을 나누는 기준은 회귀 계수가 선형/비선형에 따른 것이지 독립



과소적합 및 과적합의 이해

- 다항식의 차수가 높아질 수록 매우 복잡한 피처 간의 관계까지 모델링이 가능, 하지만 차수가 높을 수록 학습 데이터에만 맞추기에 과적합의 문제가 발생



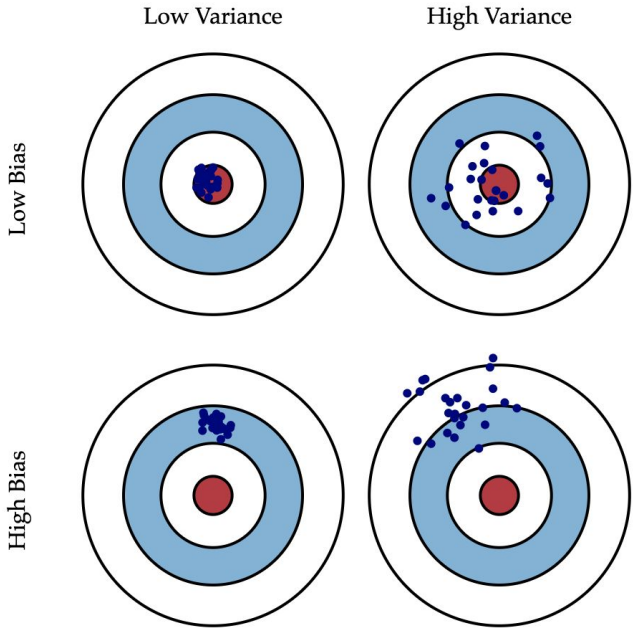
Degree:1, MSE:0.41

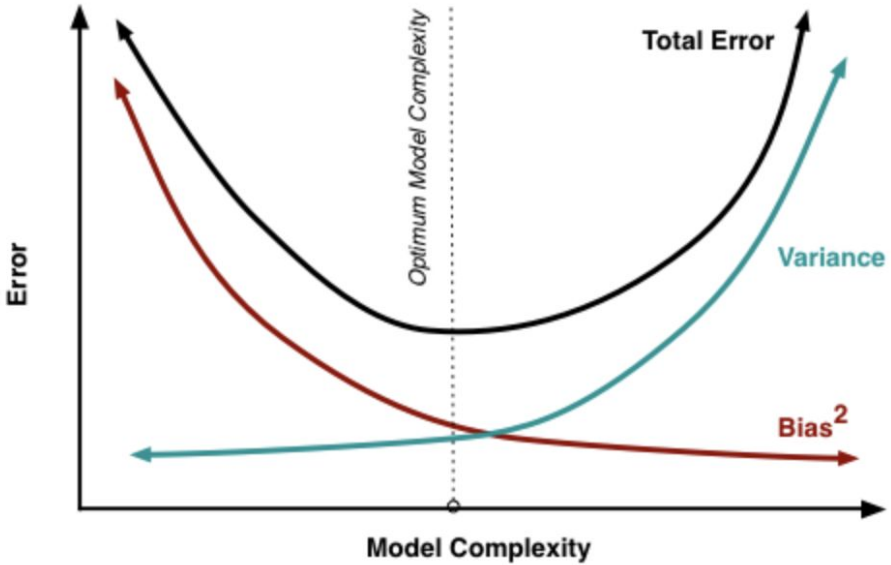
Degree:4, MSE:0.04

Degree:15, MSE:180526263

편향-분산 트레이드 오프

- 편향과 분산은 한 쪽이 높으면 한 쪽이 낮아지는 경향이 있음





출처 : http://scott.fortmann-roe.com/docs/BiasVariance.html

4. 규제 선형 모델

- 차수(Degree)가 너무 낮으면 과소적합, 너무 높으면 과대적합이 됨

비용 함수 목표 = $\mathrm{Min}(\mathrm{RSS}(\mathrm{W}) + \mathrm{alpha} * \|W\|_2^2)$

- alpha = 0 인 경우는 W가 커도 결국 0이므로 비용 함수는 Min(RSS(W))
- alpha = 무한대 인 경우는 W 를 0에 가깝게 최소화 해야 함

릿지 회귀

- L2 방식의 규제를 적용한 회귀
- 거의 영향을 미치지 않는 특성에 대하여 0에 가까운 가중치를 줌

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

# alphas list 값을 iteration하면서 alpha에 따른 평균 rmse 구함.
for alpha in alphas :
    ridge = Ridge(alpha = alpha)

    #cross_val_score를 이용하여 5 fold의 평균 RMSE 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: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

- alpha 값이 증가하면서 회귀 계수가 지속적으로 작아지지만, 0은 되지 않음

라쏘 회귀

- L1 규제 방식을 적용한 회귀
- 불필요한 회귀 계수를 0으로 제거하여, 적절한 피처만 회귀에 포함 시킴

```
# 라쏘에 사용될 alpha 파라미터의 값들을 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)
```

```
#######  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
```

	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

- alpha 값이 증가하면서 회귀 계수가 0이 됨을 알 수 있음

엘라스틱넷 회귀

- L2 규제와 L1 규제를 결합한 회귀
- 라쏘 회귀로 인해 alpha 값에 따라 회귀 계수가 0이 되는 피쳐가 천차만별일 수 있는데, 이를 완화하기 위해 L2 규제를 추가한 것
- L1 과 L2 규제가 결합되었기에 상대적으로 수행시간이 오래 걸림

```
# 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)
```

```
#######  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
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.574162	3.414154	1.918419	0.938789	0.000000
CHAS	1.330724	0.979706	0.000000	0.000000	0.000000
RAD	0.278880	0.283443	0.300761	0.289299	0.146846
ZN	0.050107	0.050617	0.052878	0.052136	0.038268
B	0.010122	0.010067	0.009114	0.008320	0.007020
AGE	-0.010116	-0.008276	0.007760	0.020348	0.043446
TAX	-0.014522	-0.014814	-0.016046	-0.016218	-0.011417
INDUS	-0.044855	-0.042719	-0.023252	-0.000000	-0.000000
CRIM	-0.099468	-0.099213	-0.089070	-0.073577	-0.019058
NOX	-0.175072	-0.000000	-0.000000	-0.000000	-0.000000
LSTAT	-0.574822	-0.587702	-0.693861	-0.760457	-0.800368
PTRATIO	-0.779498	-0.784725	-0.790969	-0.738672	-0.423065
DIS	-1.189438	-1.173647	-0.975902	-0.725174	-0.031208

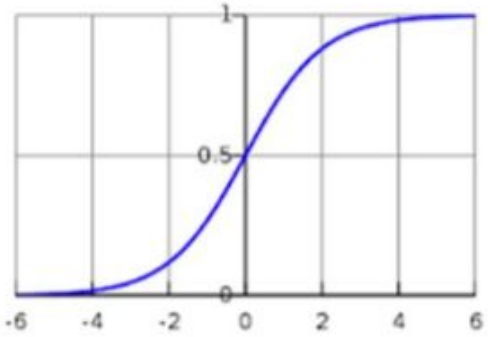
- 라쏘 회귀보단 상대적으로 0이 되는 값이 적음

- 일반적으로 선형 회귀를 적용하려는 데이터 세트에 데이터 값의 분포가 심하게 왜곡되어 있을 경우에는 로그 변환을 적용하는 것이 좋은 결과를 기대할 수 있음.

변환유형	alpha 값			
	0.1	**1**	**10**	**100**
원본데이터	5.988	5.653	5.518	5.330
표준 정규 분포	5.826	5.803	5.637	5.421
표준 정규 분포 + 2차 다항식	8.827	6.871	5.485	**4.634**
최소값/최대값 정규화	5.764	5.465	5.754	7.635
최소/최대값 정규화 + 2차 다항식	5.298	**4.323**	5.185	6.538
로그 변환	**4.770**	**4.676**	**4.836**	6.241

5. 로지스틱 회귀

- 선형 회귀 방식을 분류에 적용한 알고리즘
- 선형 회귀와 다른 점은 학습을 통해 선형 함수의 회귀 최적선을 찾는 것이 아니라 시그모이드 함수 최적선을 찾고 이 시그모이드 함수의 반환 값을 확률로 간주해 확률에 따라 분류를 결정
- 가볍고 빠르며, 이진 분류 예측 성능도 뛰어나기에 이진 분류의 기본 모델로 사용하는 경우가 많음



6. 회귀 트리

- 분류 트리와 크게 다르지 않지만, 리프 노드에서 결정 값을 만드는 과정에서 차이가 있음
- 분류 -> 특정 클래스 레이블
- 회귀 -> 데이터 값의 평균

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

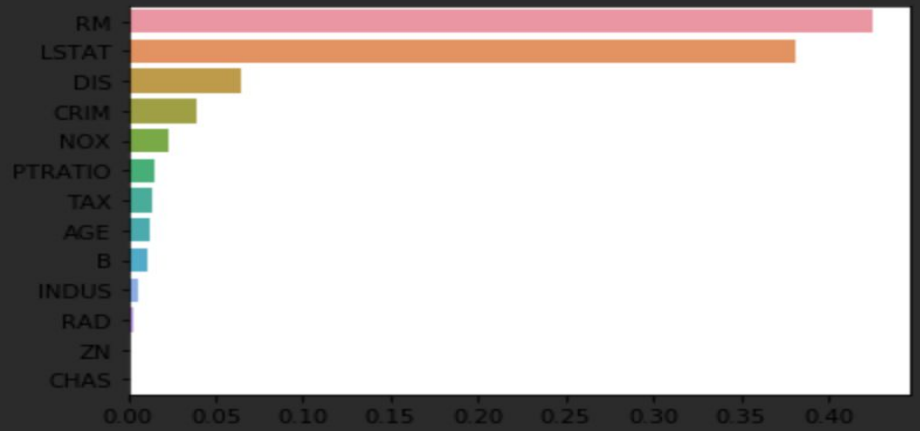
```
from sklearn.datasets import load_boston
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np

# 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:  [ -7.93 -13.06 -20.53 -46.31 -18.8 ]
 5 교차 검증의 개별 RMSE scores :  [2.82 3.61 4.53 6.8  4.34]
 5 교차 검증의 평균 RMSE : 4.420
```

- 랜덤 포레스트 회귀 트리를 통해 보스턴 주택 가격 예측

```
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)
```

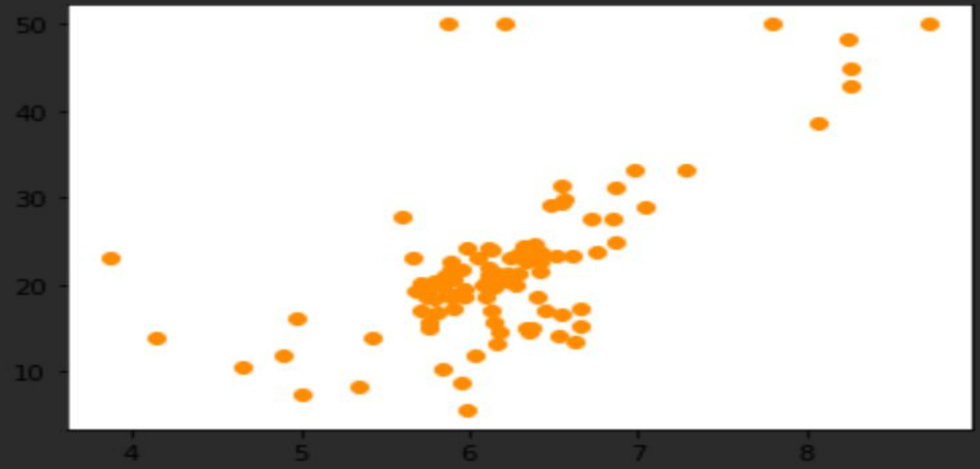
```
<matplotlib.axes._subplots.AxesSubplot at 0x10ee5fc40>
```



- feature_importance 를 통해 피쳐별 중요도를 알 수 있음

```
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")
```

```
(100, 2)
<matplotlib.collections.PathCollection at 0x124d856d0>
```



- 가장 밀접한 관계를 가졌던 RM 을 이용해 100개 샘플링

```
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)
```

```
[<matplotlib.lines.Line2D at 0x128953130>]
```