Regularized Linear Regression

# 주차	11

Loss function

머신러닝을 통해 생성한 모형의 예측값이 실제값과 얼마나 차이가 나는지 산출함

- **L1**
  - $\sum_{i=1}^{n} |y_i - \hat{y}_i|$
  - mean absolute error $\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$
- **L2**
  - $\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
  - mean squared error $\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
  - root mean squared error $\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$
  - 이상치에 더 민감하다.

Regularization

overfitting을 해결하기 위한 방법은 크게 두 가지가 있다.

1. reduce number of features

   특성의 개수를 줄이는 방법이다. 어떤 특성을 사용할 것인지 자동으로 선택하거나 제거하는 알고리즘을 활용할 수 있다.

2. regularization

   모든 특성을 유지하지만, 각 특성이 갖는 영향을 줄이는 방법이다. **손실함수(Loss function)에 가중치의 norm을 더한 함수를 목적 함수(Objective function)으로 설정하여 가중치를 제한**한다.

Ridge

$\mathrm{Loss}(x, y) = \mathrm{argmin}_W (RSS(W)) + \alpha \sum W^2$

- α를 0 또는 매우 작은 값으로 설정한다면, 손실 함수 식은 기존과 동일한 $\mathrm{Loss}(x, y) = \mathrm{argmin}_W (RSS(W)) + 0$이 될 것이다.
- α를 무한대 또는 매우 큰 값으로 설정한다면, 손실 함수 식은 $RSS(W)$에 비해 $\alpha \sum W^2$의 값이 너무 커지게 되면서 W를 작게 만들어야 손실이 최소화되는 형태가 될 것이다.
  - 즉, α 값을 크게 하면 회귀 계수 W의 값을 작게 하도록 유도할 수 있다.

LASSO

least absolute shrinkage and selection operator, Ridge와 같으나, L1을 활용해 규제

$\mathrm{Loss}(x, y) = \mathrm{argmin}_W (RSS(W)) + \alpha \sum |W|$

- 변수 선택 기능을 제공한다.

∵ 불필요한 변수에 대한 가중치를 0에 가깝게 줄이는 Ridge와 달리, 불필요한 변수에 대한 가중치를 완전히 0으로 억압할 수 있기 때문

- 미분 가능해 보다 매끄럽게 최적점에 도달하는 Ridge와 달리, 0에서 미분 가능하지 않으므로 진동하며 수렴한다.

Elastic Net

Ridge와 Lasso를 합친 형태

$$\text{Loss}(x, y) = \text{argmin}_W(RSS(W)) + \rho\alpha\sum|W| + \frac{1-\rho}{2}\alpha\sum W^2$$

- α가 0에 가까울수록 Ridge와 유사해지고, α가 1에 가까울수록 LASSO와 유사해진다.

Code

California housing

- loading data

```
from sklearn.datasets import fetch_california_housing
california = fetch_california_housing()
california
```

✓ 0.0s Python


```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.55555556,
          37.88      , -122.23      ],
       [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
          37.86      , -122.22      ],
       [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
          37.85      , -122.24      ],
       ...,
       [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
          39.43      , -121.22      ],
       [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
          39.43      , -121.32      ],
       [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
          39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset\n--------------
```


- linear regression and ridge regression cross-validation

> 💡 Cross-validation
>
> - train set을 train set과 validation set으로 나눠 1차 검증하는 것
> - k개의 fold로 나눠 그 중 1개의 fold는 validation set이 됨
> - k개의 성능 결과가 나오는데, 이들의 평균이 모델의 성능이 됨
> - 각 성능 결과는 MSE로 평가하는데, 더 낮은 점수가 더 좋은 모델을 나타내므로 음수 부호를 붙여서 반환

```python
lr = LinearRegression()
lr_neg_mse_scores = cross_val_score(lr, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
lr_rmse_scores = np.sqrt(-1*lr_neg_mse_scores)
lr_avg_rmse = np.mean(lr_rmse_scores)
```

[22] ✓ 0.0s Python

```python
ridge = Ridge(alpha=10)
neg_mse_scores = cross_val_score(ridge, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
rmse_scores = np.sqrt(-1*neg_mse_scores)
avg_rmse = np.mean(rmse_scores)
```

[23] ✓ 0.0s Python

```python
print('선형회귀: ', lr_avg_rmse)
print("릿지: ", avg_rmse)
```

[25] ✓ 0.0s Python

```
선형회귀:  0.7266657633193192
릿지:  0.7265690986274753
```

- alpha parameter 값을 조정한다면?

```python
alphas = [0, 0.1, 1, 10, 100]

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    neg_mse_scores = cross_val_score(ridge, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
    avg_mse = np.mean(np.sqrt(-1*neg_mse_scores))
    print(f'alpha {alpha}일 때 5 folds의 평균 RMSE: {avg_rmse:.3f}')
```

✓ 0.2s Python

```
alpha 0일 때 5 folds의 평균 RMSE: 0.727
alpha 0.1일 때 5 folds의 평균 RMSE: 0.727
alpha 1일 때 5 folds의 평균 RMSE: 0.727
alpha 10일 때 5 folds의 평균 RMSE: 0.727
alpha 100일 때 5 folds의 평균 RMSE: 0.727
```

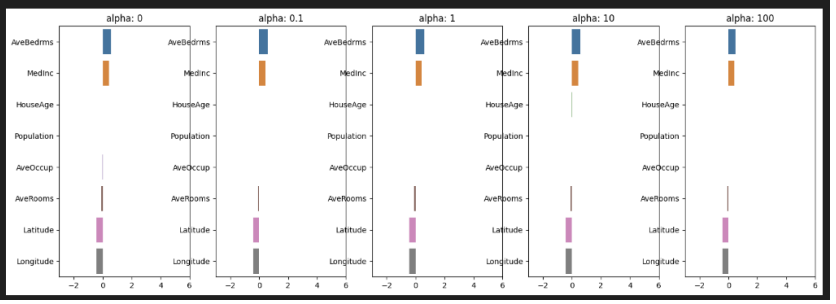
```python
# 각 alpha 값에 따른 회귀 계수 값 시각화
fig, axs = plt.subplots(figsize=(18, 6), nrows=1, ncols=5)

coeff_df = pd.DataFrame()
for pos, alpha in enumerate(alphas):
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)

    coeff = pd.Series(data=ridge.coef_, index=X_train.columns)
    colname = 'alpha: ' + str(alpha)
    coeff_df[colname] = coeff

    coeff = coeff.sort_values(ascending=False) # 회귀 계수 값이 높은 것부터 표현
    axs[pos].set_title(colname)
    axs[pos].set_xlim(-3, 6)
    sns.barplot(x=coeff.values, y=coeff.index, ax=axs[pos])
```

✓ 0.7s Pyt



- LASSO와 ElasticNet도 evaluate 하고 싶다면?

```python
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)
        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(f'alpha {param}일 때 5 fold set의 평균 RMSE: {avg_rmse:.3f}')

        model.fit(X_data_n, y_target_n) #cross_val_score은 evaluation metric만 반환하므로 다시 훈련
        if return_coeff:
            coeff = pd.Series(data=model.coef_, index=X_data_n.columns)
            colname = 'alpha' + str(param)
            coeff_df[colname] = colname
    return coeff_df
```

✓ 0.0s Python

```python
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_train, y_target_n=y_train)
```

✓ 0.3s Python

```
#### Lasso ####
alpha 0.07일 때 5 fold set의 평균 RMSE: 0.755
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.777
alpha 0.5일 때 5 fold set의 평균 RMSE: 0.850
alpha 1일 때 5 fold set의 평균 RMSE: 0.972
alpha 3일 때 5 fold set의 평균 RMSE: 1.156
```

```python
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_train, y_target_n=y
```

✓ 0.2s Python

```
#### ElasticNet ####
alpha 0.07일 때 5 fold set의 평균 RMSE: 0.742
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.750
alpha 0.5일 때 5 fold set의 평균 RMSE: 0.825
alpha 1일 때 5 fold set의 평균 RMSE: 0.874
alpha 3일 때 5 fold set의 평균 RMSE: 1.149
```

- scaling data

```python
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
```

✓ 0.0s Python

  - Standard Scaler

    $$X_{\text{scaled}} = \frac{X-\mu}{\sigma}$$

  - MinMax Scaler

    $$X_{\text{scaled}} = \frac{X-X_{\min}}{X_{\max}-X_{\min}}$$

  - Log Scaler

    $$X_{\text{scaled}} = \log(1+X)$$

▼ `p_degree`는 모델이 비선형관계도 학습할 수 있도록 다항식 변환을 수행한다.

예시

원본 데이터

python 코드 복사

```python
import numpy as np

# 예제 데이터
X = np.array([[2, 3], [3, 4], [4, 5]])
```

다항식 변환 전

- 원본 데이터 `X`:

$$\begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix}$$

다항식 변환 후 (`degree=2`)

python 코드 복사

```python
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)
print(X_poly)
```

출력:

$$\begin{bmatrix} 2 & 3 & 4 & 9 & 6 \\ 3 & 4 & 9 & 16 & 12 \\ 4 & 5 & 16 & 25 & 20 \end{bmatrix}$$

설명

- 원본 데이터의 각 특성에 대해 2차 다항식 조합이 생성됩니다.
- 예를 들어, 첫 번째 샘플 `[2, 3]`에 대해:
  - 원본 특성: `2, 3`
  - 2차 다항식 조합: `2^2, 3^2, 2*3`
  - 최종 결과: `[2, 3, 4, 9, 6]`

```python
scale_methods = [(None, None), ('Standard', None), ('Standard', 2), ('MinMax',None),('MinMax',2)]
alphas = [0.1, 1, 10, 100]

for scale_method in scale_methods:
    X_data_scaled = get_scaled_data(method=scale_method[0], p_degree=scale_method[1], input_data=X_train)
    print(f'변환 유형: {scale_method[0]}, Polynomial Degree: {scale_method[1]}')
    get_linear_reg_eval('Ridge', params=alphas, X_data_n=X_data_scaled,
                        y_target_n=y_train, verbose=False, return_coeff=False)
```

✓ 1.1s Python

```
변환 유형: None, Polynomial Degree: None
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.727
alpha 1일 때 5 fold set의 평균 RMSE: 0.727
alpha 10일 때 5 fold set의 평균 RMSE: 0.727
alpha 100일 때 5 fold set의 평균 RMSE: 0.726
변환 유형: Standard, Polynomial Degree: None
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.727
alpha 1일 때 5 fold set의 평균 RMSE: 0.727
alpha 10일 때 5 fold set의 평균 RMSE: 0.727
alpha 100일 때 5 fold set의 평균 RMSE: 0.727
변환 유형: Standard, Polynomial Degree: 2
alpha 0.1일 때 5 fold set의 평균 RMSE: 1.611
alpha 1일 때 5 fold set의 평균 RMSE: 1.574
alpha 10일 때 5 fold set의 평균 RMSE: 1.329
alpha 100일 때 5 fold set의 평균 RMSE: 0.838
변환 유형: MinMax, Polynomial Degree: None
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.726
alpha 1일 때 5 fold set의 평균 RMSE: 0.729
alpha 10일 때 5 fold set의 평균 RMSE: 0.735
alpha 100일 때 5 fold set의 평균 RMSE: 0.802
변환 유형: MinMax, Polynomial Degree: 2
alpha 0.1일 때 5 fold set의 평균 RMSE: 0.696
alpha 1일 때 5 fold set의 평균 RMSE: 0.711
alpha 10일 때 5 fold set의 평균 RMSE: 0.727
alpha 100일 때 5 fold set의 평균 RMSE: 0.768
```

- visualization

```python
def plot_predictions(y_test, y_pred, title, alpha=None):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.3)
    plt.plot([0, 5], [0, 5], '--r')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    if alpha is not None:
        plt.title(f'{title} (alpha={alpha})')
    else:
        plt.title(title)
    plt.show()
```

✓ 0.0s Python

  - linear regression

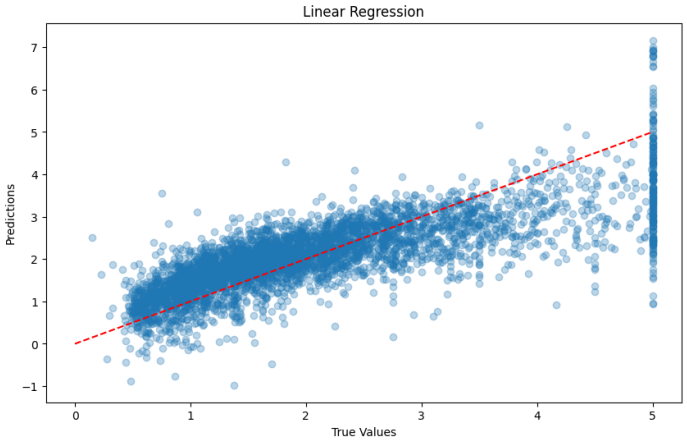
```python
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Linear Regression MSE: {mse:.4f}')
```

✓ 0.0s Python

```
Linear Regression MSE: 0.5290
```

```python
plot_predictions(y_test, y_pred, 'Linear Regression')
```

✓ 0.2s Python



  - different models

```python
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
ridge_mse = []
ridge_predictions = {}

for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    ridge_mse.append(mse)
    ridge_predictions[alpha] = y_pred
```

```python
lasso_mse = []
lasso_predictions = {}

for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    lasso_mse.append(mse)
    lasso_predictions[alpha] = y_pred
```

```python
elasticnet_mse = []
elasticnet_predictions = {}
l1_ratio = 0.5 # L1과 L2의 비율을 고정

for alpha in alphas:
    elasticnet = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)
    elasticnet.fit(X_train, y_train)
    y_pred = elasticnet.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    elasticnet_mse.append(mse)
    elasticnet_predictions[alpha] = y_pred
```

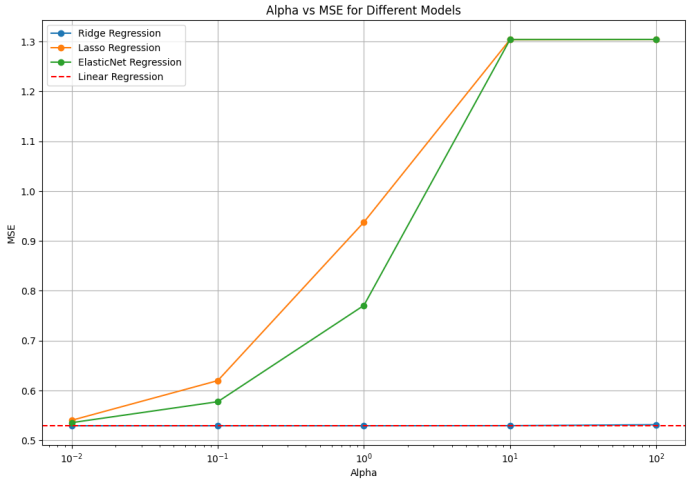
```python
plt.figure(figsize=(12, 8))

plt.plot(alphas, ridge_mse, marker='o', label='Ridge Regression')
plt.plot(alphas, lasso_mse, marker='o', label='Lasso Regression')
plt.plot(alphas, elasticnet_mse, marker='o', label='ElasticNet Regression')

plt.axhline(y=lr_mse, color='r', linestyle='--', label='Linear Regression')

plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.title('Alpha vs MSE for Different Models')
plt.legend()
plt.grid(True)
plt.show()
```

0.4s Python



- 가장 좋은 결과를 낸 alpha 값 찾기

```python
best_alpha_ridge = alphas[np.argmin(ridge_mse)]
best_alpha_lasso = alphas[np.argmin(lasso_mse)]
best_alpha_elasticnet = alphas[np.argmin(elasticnet_mse)]

print(best_alpha_ridge)
print(best_alpha_lasso)
print(best_alpha_elasticnet)
```

✓ 0.0s Python

```
0.01
0.01
0.01
```

- 모형의 최종 성능 비교하기

```python
models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'Elastic Net Regression']
mse_values = [lr_mse, np.min(ridge_mse), np.min(lasso_mse), np.min(elasticnet_mse)]

plt.figure(figsize=(10, 6))
plt.bar(models, mse_values)
plt.xlabel('models')
plt.ylabel('mse')
plt.title('FIanl Model Comparison')
plt.show()
```

✓ 0.1s Python