회귀 기법
Regression Techniques

이건명
충북대학교 소프트웨어학과

학습 내용

- 선형 회귀를 위한 학습 알고리즘으로 경사 하강법에 대해서 알아본다. .
- 선형 회귀의 목적함수를 변형한 리지 회귀, 라소 회귀, 일래스틱 회귀에 대해서 알아본다.
- 이상치가 많은 데이터에 대한 회귀 기법인 RANSAC에 대해서 알아본다.
- 단조 함수 형태를 근사하는 이소토닉 회귀에 대해서 알아본다.
- 비선형인 형태의 함수를 근사하는 다항 회귀에 대해서 알아본다.
- 서포트 벡터 머신의 개념을 적용한 서포트 벡터 회귀에 대해서 알아본다.

1. 선형 회귀

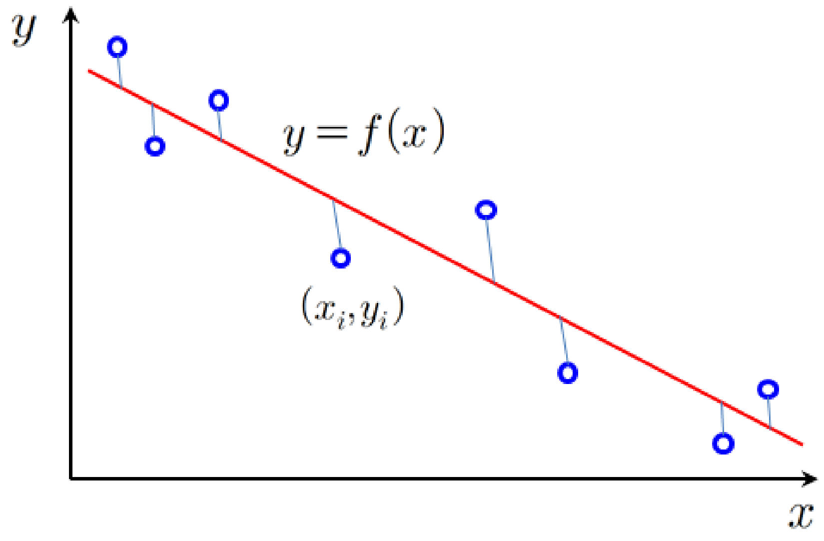
❖ **회귀** (regression)

- **학습 데이터에 부합**되는 **출력**값이 **실수**인 함수를 찾는 문제

❖ **선형회귀**(linear regression)

- 선형 함수 $f(x) = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_m x_n$을 사용한 함수 근사

$$f^*(x) = \arg\min_f \sum_{i=1}^{n} (\boldsymbol{y_i} - \boldsymbol{f}(\boldsymbol{x_i}))^2$$

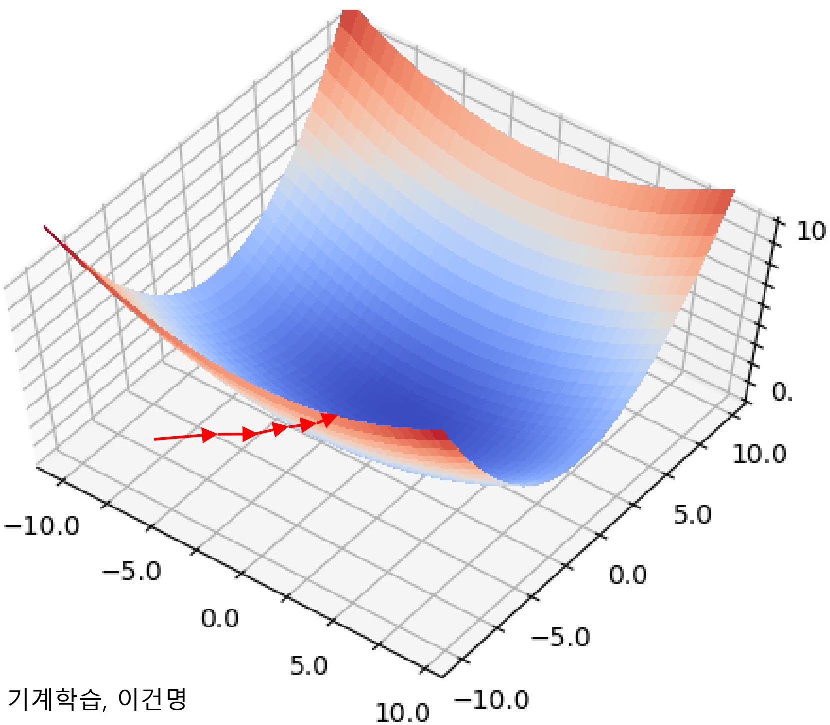


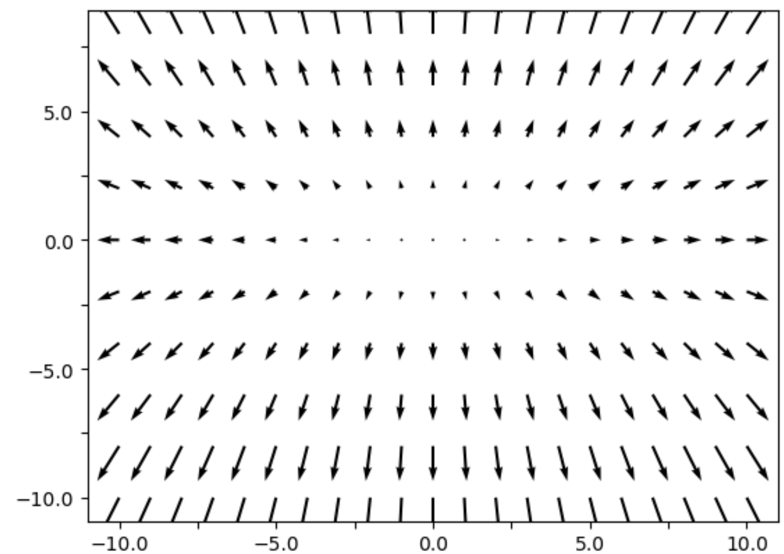
선형회귀

❖ **학습방법**

- **경사 하강법**
  - **학습 데이터에 부합**되는 **출력**값이 되도록 **파라미터 변경**하는 일

$$E = \frac{1}{2}\sum_i (y_i - f(\boldsymbol{x}_i))^2 \qquad \nabla E = \left(\frac{\partial E}{\partial a}, \frac{\partial E}{\partial b}\right)$$





$$a^{(t+1)} \leftarrow a^{(t)} - \eta\nabla_a$$

선형회귀

❖ 학습방법

- 선형회귀의 경사 하강법

$$E = \frac{1}{2}\sum_i (y_i - f(\boldsymbol{x}_i))^2$$

$$f(\boldsymbol{x}_i) = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_n x_{in}$$

$$\frac{\partial E}{\partial w_k} = -\sum_i \left(y_i - f(\boldsymbol{x}_i)\right) x_{ik}$$

$$\frac{\partial E}{\partial w_0} = -\sum_i \left(y_i - f(\boldsymbol{x}_i)\right)$$

```
def fit(self, X, y):
        self.w_ = np.zeros(1+X.shape[1])
        self.cost_ = [ ]

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta*X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum()/2.0
            self.cost_.append(cost)
        return self
```

$$w_k \leftarrow w_k - \eta \frac{\partial E}{\partial w_k} \qquad w_0 \leftarrow w_0 - \eta \frac{\partial E}{\partial w_0}$$

❖ [실습] 경사하강법에 의한 선형회귀 학습

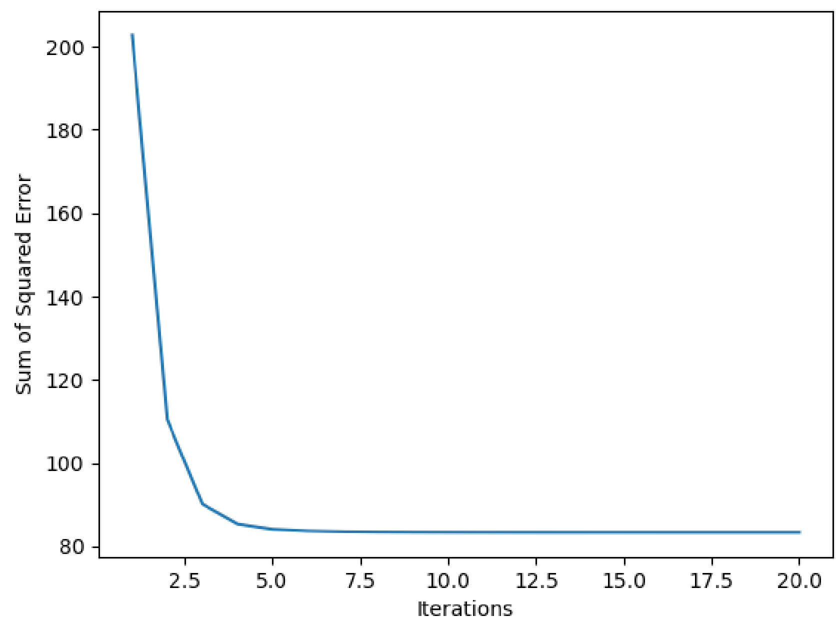
```python
import numpy as np

class LinearRegressionGD(object):
    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1+X.shape[1])
        self.cost_ = [ ]

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta*X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum()/2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)
```

```python
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

boston = load_boston()
data = pd.DataFrame(boston.data,
                    columns=boston.feature_names)
print(data.head())
print(y[0:5])

X = data[['RM', 'PTRATIO']].values
y = boston.target
```

```
      CRIM    ZN  INDUS  CHAS    NOX  ...  RAD    TAX  PTRATIO       B  LSTAT
0  0.00632  18.0   2.31   0.0  0.538  ...  1.0  296.0     15.3  396.90   4.98
1  0.02731   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  396.90   9.14
2  0.02729   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  392.83   4.03
3  0.03237   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  394.63   2.94
4  0.06905   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  396.90   5.33
```

```
[24.  21.6 34.7 33.4 36.2]
```

```
sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = sc_y.fit_transform(y[:,np.newaxis]).flatten()
X_train, X_test, y_train, y_test = train_test_split(X_std, y_std, test_size=0.2, random_state=123)

lr = LinearRegressionGD( )
lr.fit(X_train, y_train)

import matplotlib.pyplot as plt
plt.plot(range(1, lr.n_iter+1), lr.cost_)
plt.ylabel('Sum of Squared Error')
plt.xlabel('Iterations')
plt.show()

from sklearn.metrics import mean_squared_error
preds = lr.predict(X_test)
mse = mean_squared_error(y_test,preds)
print('Root mean squared error : ', np.sqrt(mse))
```



Root mean squared error : 0.7389665258932069

2. 리지 회귀

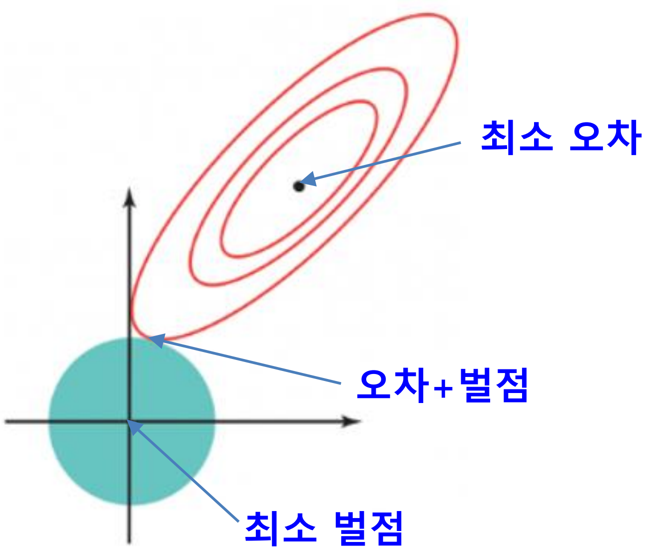
❖ **리지 회귀**(ridge regression, **능형회귀**)

- 가중치의 L2-노름을 벌점을 손실함수에 추가

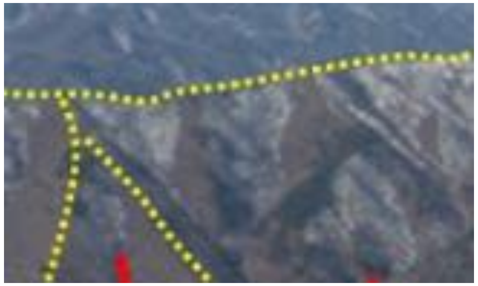
$$\boxed{L2 : \|\boldsymbol{w}\|_2^2 = \sum_{i=1}^{n} w_i^2}$$

$$\text{목적함수} = \text{MSE}(\boldsymbol{w}) + \alpha \frac{1}{2} \|\boldsymbol{w}\|_2^2$$

- 과적합 회피에 도움



Ridge (능선)



$$f(\boldsymbol{x}_i) = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_n x_{in}$$

$$w = (w_0, w_1, \cdots w_n)$$

3. 라소 회귀

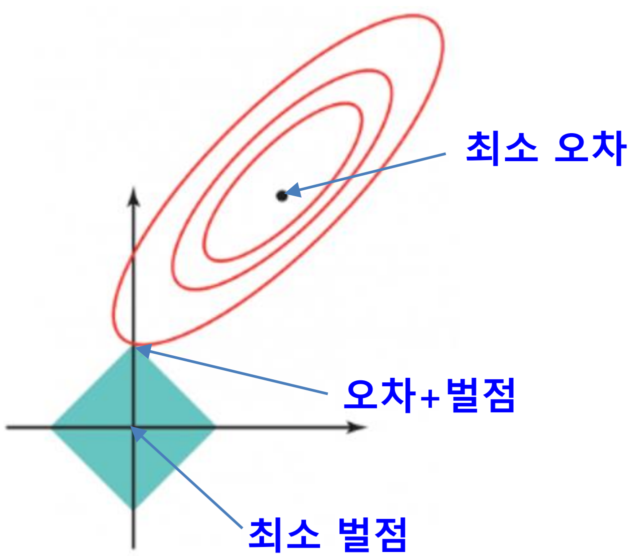
❖ **라소 회귀**(Lasso regression)

- 가중치 L1-노름을 벌점으로 손실함수에 추가

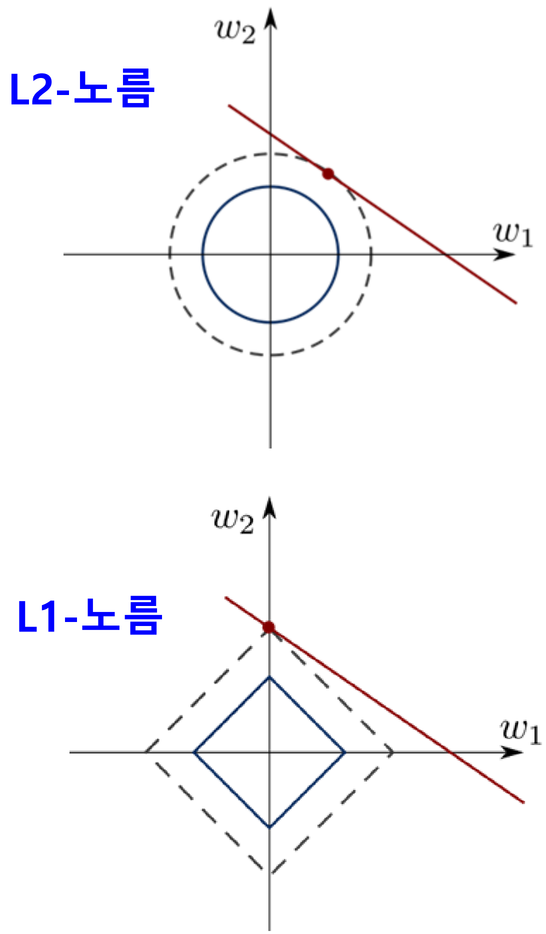
$$\boxed{L1 : \|\boldsymbol{w}\|_1 = \sum_{i=1}^{n} |w_i|}$$

$$\text{목적함수} = \mathrm{MSE}(\boldsymbol{w}) + \alpha\|\boldsymbol{w}\|_1$$

- 리지 회귀보다 단순한 모델 생성 가능



Lasso (least absolute shrinkage and selection operator)

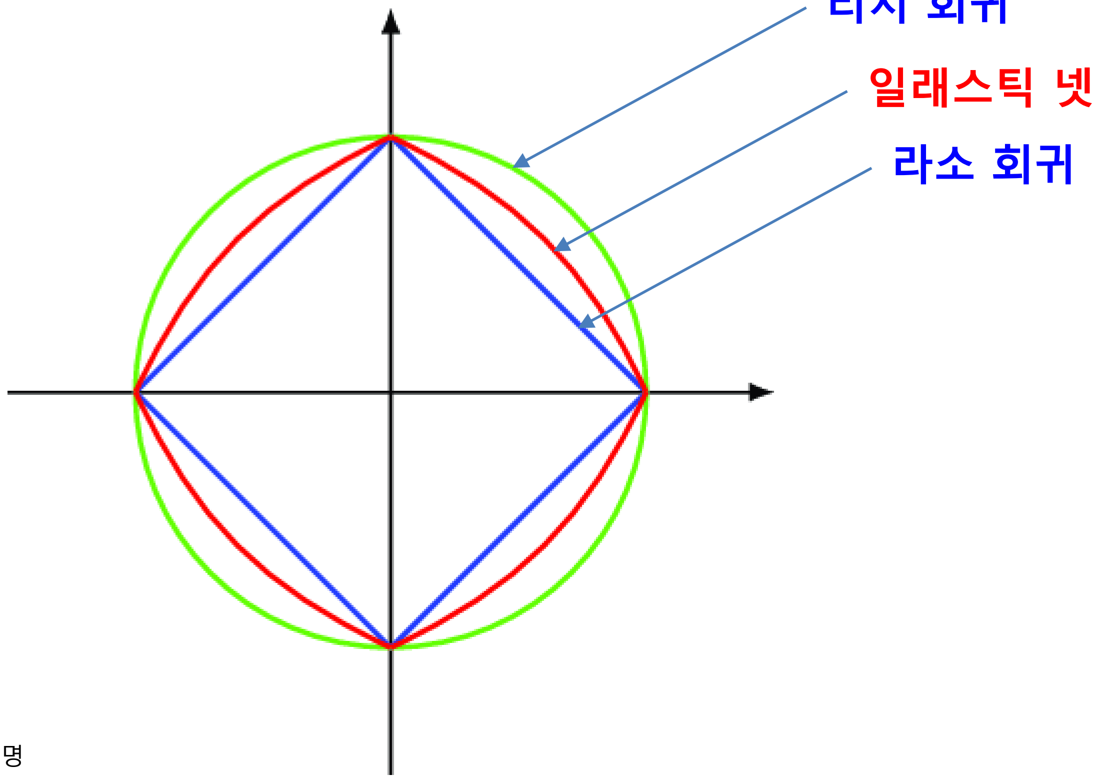


4. 일래스틱 넷

❖ **일래스틱 넷**(Elastic Net)

- 가중치의 L1-노름과 L2-노름을 벌점으로 손실함수에 추가

$$목적함수 = \mathrm{MSE}(\boldsymbol{w}) + \gamma\alpha\|\boldsymbol{w}\|_1 + \frac{1-\gamma}{2}\alpha\|\boldsymbol{w}\|_2^2$$



❖ [실습] 선형회귀, 리지 회귀, 라소 회귀, 일레스틱 넷 기반 회귀

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot  as plt
from sklearn.linear_model  import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error

boston = load_boston( )
data = pd.DataFrame(boston.data,columns=boston.feature_names)
X = data[['RM', 'PTRATIO', 'RAD', 'TAX', 'LSTAT', 'CRIM', 'NOX', 'B']].values
y = boston.target

sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = sc_y.fit_transform(y[:,np.newaxis]).flatten()
X_train, X_test, y_train, y_test = train_test_split(X_std, y_std, test_size=0.2, random_state=123)

linear = LinearRegression()
ridge = Ridge(alpha=1.0, random_state=0)
lasso = Lasso(alpha=1.0, random_state=0)
enet = ElasticNet(alpha=1.0, l1_ratio=0.5)

linear.fit(X_train, y_train)
ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)
enet.fit(X_train, y_train)
```

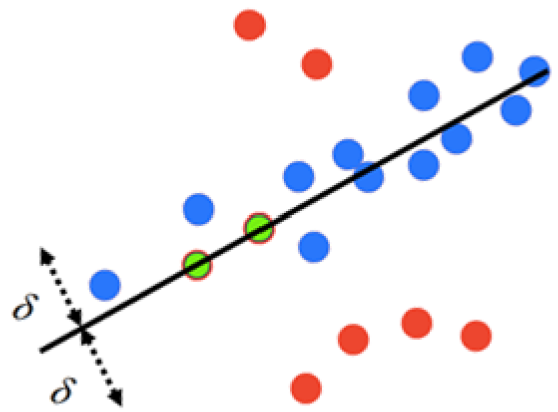
```
linear_pred = linear.predict(X_train)
ridge_pred = ridge.predict(X_train)
lasso_pred = lasso.predict(X_train)
enet_pred = enet.predict(X_train)
print('Linear  - RMSE for training data: ', np.sqrt(mean_squared_error(y_train, linear_pred)))
print('Ridge  - RMSE for training data: ', np.sqrt(mean_squared_error(y_train, ridge_pred)))
print('Lasso  - RMSE for training data: ', np.sqrt(mean_squared_error(y_train, lasso_pred)))
print('Elastic Net  - RMSE for training data: ', np.sqrt(mean_squared_error(y_train, enet_pred)))

linear_pred = linear.predict(X_test)
ridge_pred = ridge.predict(X_test)
lasso_pred = lasso.predict(X_test)
enet_pred = enet.predict(X_test)

print('₩nLinear  - RMSE for test data: ', np.sqrt(mean_squared_error(y_test, linear_pred)))
print('Ridge  - RMSE for test data: ', np.sqrt(mean_squared_error(y_test, ridge_pred)))
print('Lasso  - RMSE for test data: ', np.sqrt(mean_squared_error(y_test, lasso_pred)))
print('Elastic Net  - RMSE for test data: ', np.sqrt(mean_squared_error(y_test, enet_pred)))
```

```
Linear  - RMSE for training data:  0.527646292874511
Ridge  - RMSE for training data:  0.527652437617983l
Lasso  - RMSE for training data:  1.0017823143577615
Elastic Net  - RMSE for training data:  0.8463715147933433

Linear  - RMSE for test data:  0.6299606779044659
Ridge  - RMSE for test data:  0.6299657952065147
Lasso  - RMSE for test data:  0.9936481587307943
Elastic Net  - RMSE for test data:  0.8578764629897222
```

5. RANSAC 방법

❖ **RANSAC(RANdom Sample Consensus) 방법**

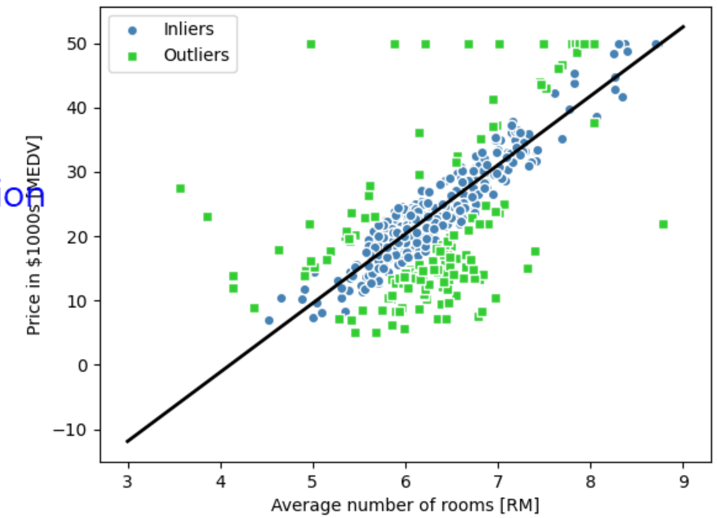
- 데이터를 **인라이어**(inlier, **정상치**)와 **아웃라이어**(outlier, **이상치**)로 구분해 회귀 과정을 반복 수행



1. 무작위로 **일부 데이터**를 **정상치**로 선택하여 모델 학습
2. 학습된 모델에 대해 다른 모든 데이터 테스트. **허용오차 이내의 데이터**를 정상치로 추가
3. 모든 정상치 데이터를 사용하여 **모델 재학습**
4. 학습된 모델에 대한 **정상치 데이터**의 **오차 계산**
5. 오차가 임계값 이내이거나 지정된 반복회수에 도달하면 종료. 아니면 단계 1로 돌아감

❖ [실습] RANSAC 기반 선형회귀

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
from sklearn.linear_model import RANSACRegressor, LinearRegression

boston = load_boston( )
data = pd.DataFrame(boston.data,columns=boston.feature_names)
print(data.head( ))
X = data[['RM']].values
y = boston.target
```

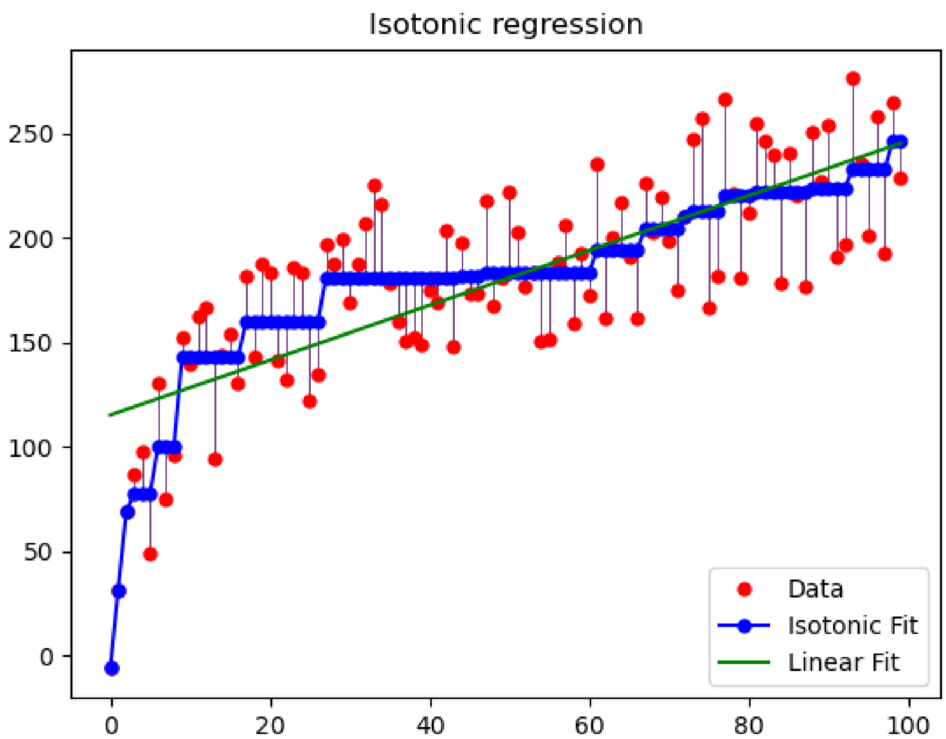



```
ransac = RANSACRegressor(LinearRegression( ), max_trials=100, min_samples=50,
                         loss='absolute_loss', residual_threshold=5.0, random_state=0)
ransac.fit(X,y)

inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
line_X = np.arange(3,10,1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])

plt.scatter(X[inlier_mask], y[inlier_mask], c='steelblue', edgecolor='white', marker='o', label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask], c='limegreen', edgecolor='white', marker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='black', lw=2)
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000s [MEDV]')
plt.legend(loc='upper left')
plt.show( )
```

6. 이소토닉 회귀

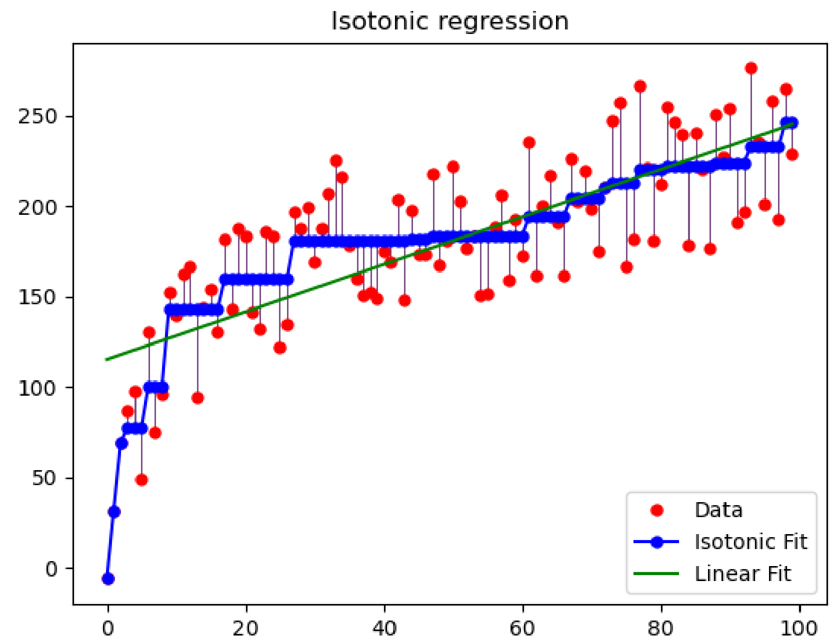
❖ 이소토닉 회귀(Isotonic Regression)

- 비감소(non-decreasing) 또는 비증가(non-increasing) 함수 회귀
- 상세한 변화 표현
- 대상 함수를 최소화하는 부분 보간 함수 사용



❖ [실습] 이소토닉 회귀

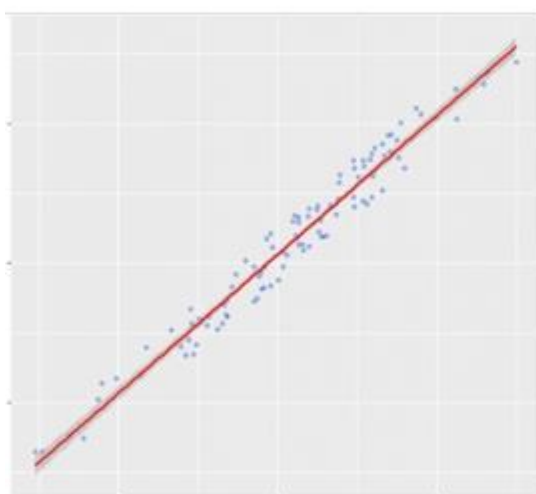
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
from sklearn.linear_model import LinearRegression
from sklearn.isotonic import IsotonicRegression
from sklearn.utils import check_random_state

n = 100
x = np.arange(n)
rs = check_random_state(0)
y = rs.randint(-50, 50, size=(n,)) + 50. * np.log1p(np.arange(n))

ir = IsotonicRegression( )
y_ = ir.fit_transform(x, y)
lr = LinearRegression( )
lr.fit(x[:, np.newaxis], y)

segments = [[[i, y[i]], [i, y_[i]]] for i in range(n)]
lc = LineCollection(segments, zorder=0)
lc.set_array(np.ones(len(y)))
lc.set_linewidths(np.full(n, 0.5))

fig = plt.figure()
plt.plot(x, y, 'r.', markersize=10)
plt.plot(x, y_, 'b.-', markersize=10)
plt.plot(x, lr.predict(x[:, np.newaxis]), g-')
plt.gca().add_collection(lc)
plt.legend(('Data', 'Isotonic Fit', 'Linear Fit'), loc='lower right')
plt.title('Isotonic regression')
plt.show()
```

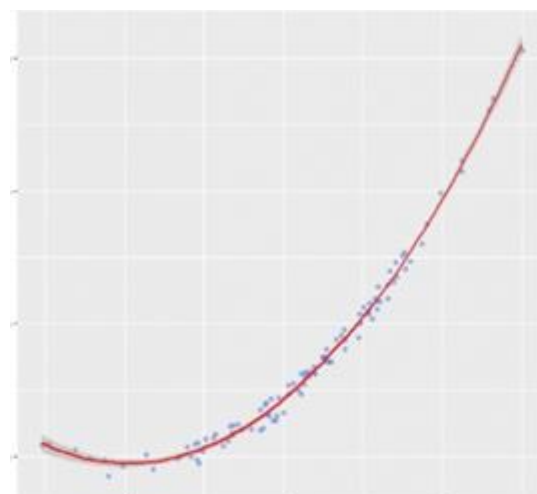


7. 다항회귀

❖ **다항회귀(polynomial regression)**

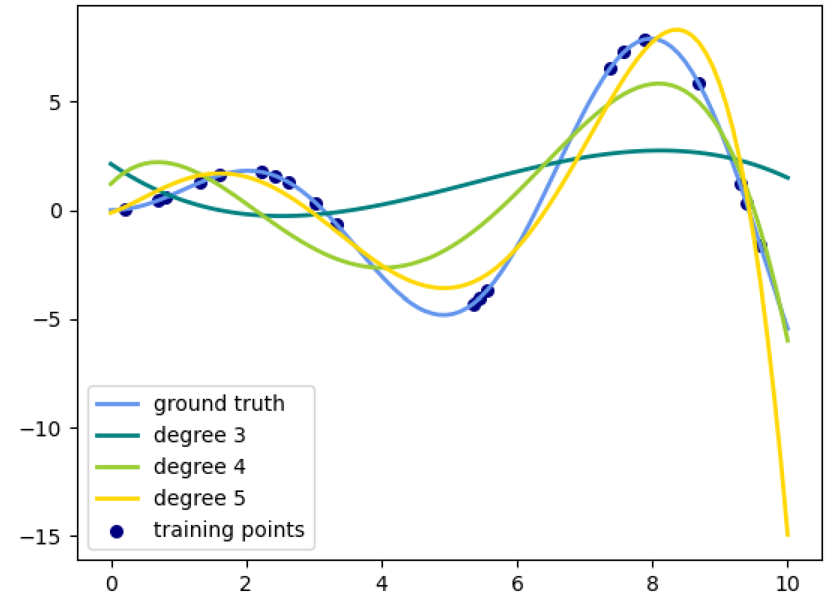
- 기존 변수들을 이용하여 계산한 여분의 변수를 추가한 선형회귀
- 비선형 형태의 함수 근사
- 예.
  - $\boldsymbol{x} = (x_1, x_2) \quad \rightarrow \quad \boldsymbol{x}' = (x_1, x_2, x_1 x_2, x_1^2, x_2^2)$





❖ [실습] 다항 회귀

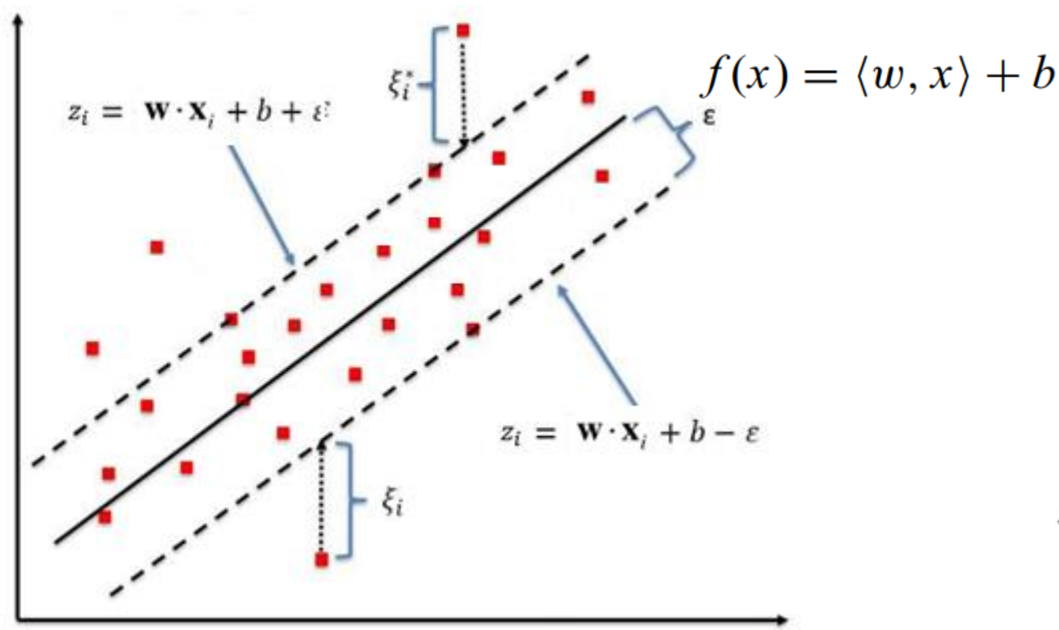
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

def f(x):
    return x * np.sin(x)

x_plot = np.linspace(0, 10, 100)
x = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
rng.shuffle(x)
x = np.sort(x[:20])
y = f(x)
X = x[:, np.newaxis]
X_plot = x_plot[:, np.newaxis]

colors = ['teal', 'yellowgreen', 'gold']
lw = 2
plt.plot(x_plot, f(x_plot), color='cornflowerblue', linewidth=lw, label="ground truth")
plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")

for count, degree in enumerate([3, 4, 5]):
    model = make_pipeline(PolynomialFeatures(degree), Ridge( ))
    model.fit(X, y)
    y_plot = model.predict(X_plot)
    plt.plot(x_plot, y_plot, color=colors[count], linewidth=lw, label="degree %d" % degree)

plt.legend(loc='lower left')
plt.show( )
```

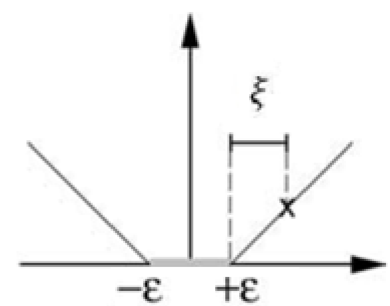


8. 서포트 벡터 회귀

❖ **서포트 벡터 회귀(SVR,** Support Vector Regression**)**

- 예측값이 목푯값을 중심으로 반지름이 엡실론(ε, 기본값 0.1)인 볼(ball)에 포함되면 패널티 미적용
- 학습 데이터 : $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_l, y)\}$





차이 $y_i - \boldsymbol{w}_i \cdot \boldsymbol{x}_i$가 ε이내이면, 오차가 없는 것으로 간주

Support Vector Regression

❖ **서포트 벡터 회귀(SVR)**

- 예측값이 목푯값을 중심으로 반지름이 엡실론(ε, 기본값 0.1)인 볼(ball)에 포함되면 패널티 미적용
- 학습 데이터 : $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_l, y)\}$

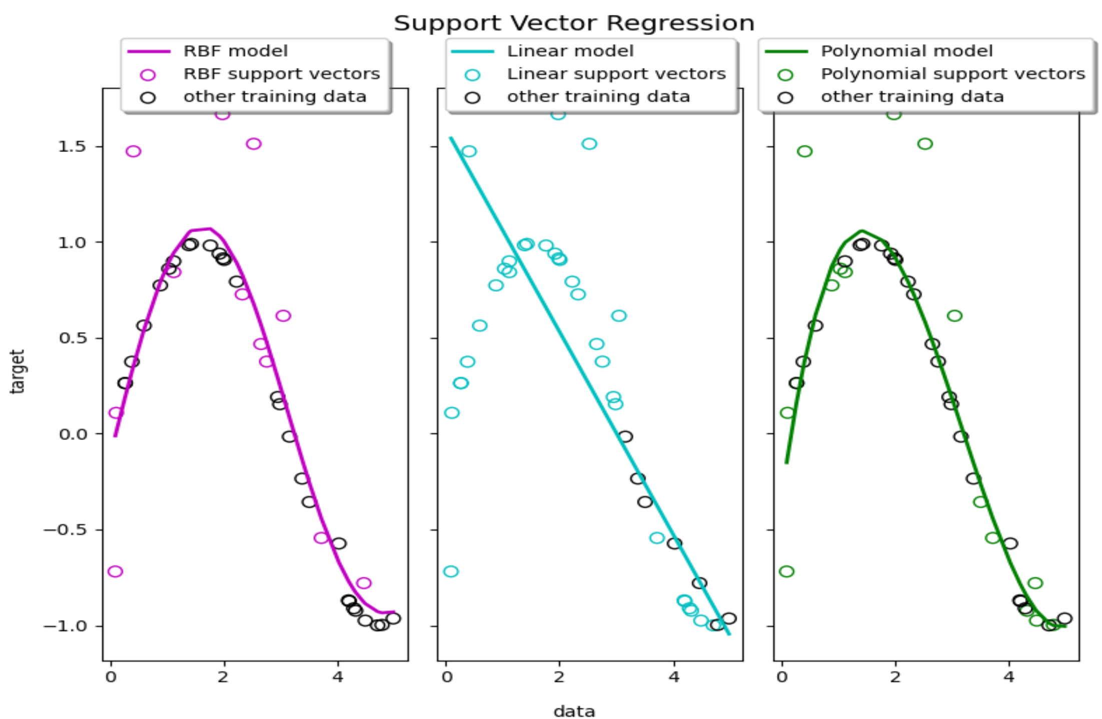
$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*) \\ \text{subject to} & \begin{cases} y_i - \langle w, x_i \rangle - b \le \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \qquad\qquad\quad \ge 0 \end{cases} \end{array}$$

$$\begin{aligned} L := \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*) - \sum_{i=1}^{\ell}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ - \sum_{i=1}^{\ell}\alpha_i(\varepsilon + \xi_i - y_i + \langle w, x_i \rangle + b) \\ - \sum_{i=1}^{\ell}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w, x_i \rangle - b) \end{aligned}$$

❖ [실습] SVR 적용 회귀

```
from sklearn.svm import SVR
import matplotlib.pyplot as plt
import numpy as np

X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel( )
y[::5] += 3 * (0.5 - np.random.rand(8))  # 잡음 추가

svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_lin = SVR(kernel='linear', C=100, gamma='auto')
svr_poly = SVR(kernel='poly', C=100, gamma='auto', degree=3, epsilon=.1, coef0=1)


lw = 2
svrs = [svr_rbf, svr_lin, svr_poly]
kernel_label = ['RBF', 'Linear', 'Polynomial']
model_color = ['m', 'c', 'g']
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(9, 7), sharey=True)
for ix, svr in enumerate(svrs):
    axes[ix].plot(X, svr.fit(X, y).predict(X), color=model_color[ix], lw=lw,
                  label='{ } model'.format(kernel_label[ix]))
    axes[ix].scatter(X[svr.support_], y[svr.support_], facecolor="none",
                     edgecolor=model_color[ix], s=50, label='{} support vectors'.format(kernel_label[ix]))
    axes[ix].scatter(X[np.setdiff1d(np.arange(len(X)), svr.support_)],
                     y[np.setdiff1d(np.arange(len(X)), svr.support_)],
                     facecolor="none", edgecolor="k", s=50, label='other training data')
    axes[ix].legend(loc='upper center', bbox_to_anchor=(0.5, 1.1), ncol=1, fancybox=True, shadow=True)

fig.text(0.5, 0.04, 'data', ha='center', va='center')
fig.text(0.06, 0.5, 'target', ha='center', va='center', rotation='vertical')
fig.suptitle("Support Vector Regression", fontsize=14)
plt.show( )
```



Quiz

1. 리지 회귀가 라소 회귀보다 더 단순한 모델을 학습할 가능성이 높다. (O,X)
2. RANSAC은 이상치가 많은 데이터 집합에 대한 회귀에 적용할 수 있다. (O,X)
3. SVR은 목표 함수에서 일정 거리 이내 만큼 떨어진 데이터에 대해서는 오차가 없는 것으로 간주한다. (O,X)
4. 선형 회귀를 통해서는 비선형인 형태의 함수를 학습할 수 없다. (O,X)
5. 이소토닉 회귀를 사용하면 선형 회귀에서 보다 지역적인 변화를 민감하게 확인할 수 있다. (O,X)
6. 일래스틱넷 모델은 항상 리지 회귀보다 좋은 결과를 보여준다. (O,X)