머신 러닝·딥러닝에 필요한 수학 기초 with 파이썬

Linear Regression

선형 회귀

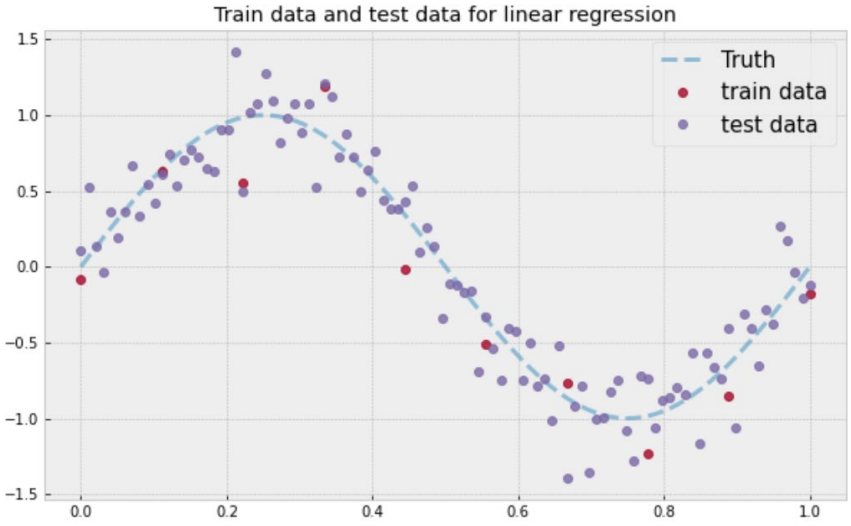
조준우
metamath@gmail.com

학습 개요

- 선형회귀
  - 목적함수의 정의
  - 다항 기저함수
  - 목적함수의 미분
  - 경사 하강법 적용
  - 정규방정식
- 머신러닝 일반 이슈
  - 과적합과 레귤러라이제이션
- 다변수 선형회귀
- 주요 참고 문헌
  - Pattern Recognition and Machine Learning, Christopher Bishop, Springer

학습데이터 생성

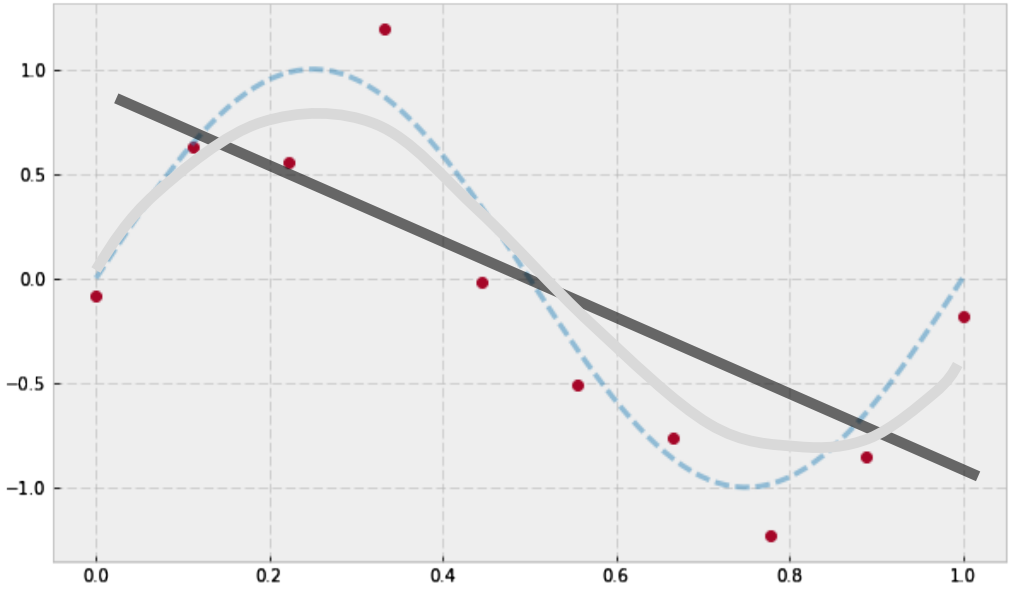
- Sin 함수 + 적당한 노이즈




```
N=10
x_train = np.linspace(0, 1, N)
t_train =
np.sin(2*np.pi*x_train) +
(np.random.randn(N)/5)
```

```
N_test=100
x_test = np.linspace(0, 1,
N_test)
t_test =
np.sin(2*np.pi*x_test) +
(np.random.randn(N_test)/5)
```

학습데이터와 모델



빨간 점만 보고

곡선을 찾고 싶다.

휘지 않는 막대를 쓸까?

휘어지는 철사를 쓸까?

MODEL

선형 모델Linear Model

- 주어진 데이터를 직선으로 표현

$$y(x, \mathbf{w}) = w_0 + w_1 x$$

- 입력 x는 스칼라(숫자 하나) 결정해야 하는 파라미터 2개 w_0, w_1

$$y(x, \mathbf{w}) = w_0 \times 1 + w_1 \times x$$

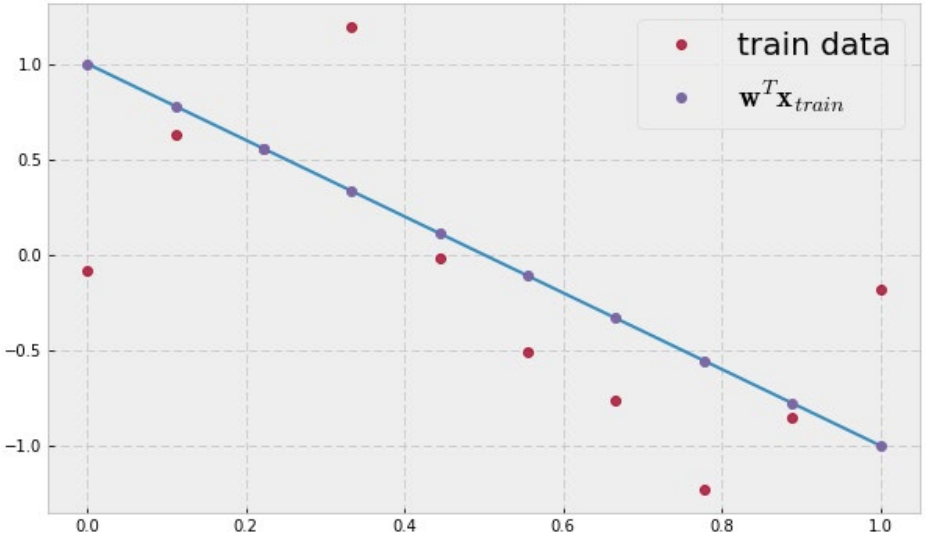
- 벡터 형식

$$y(\mathbf{x}, \mathbf{w}) = w_0 \times 1 + w_1 \times x = (w_0 \quad w_1) \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

선형 모델

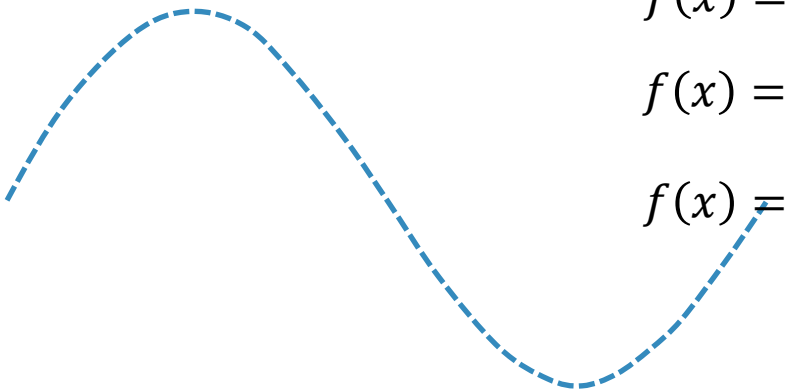
- $w_0 = 1, w_1 = -2$라면 $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ 는 아래와 같다.
- 여기서 $\mathbf{x_n} = \begin{pmatrix} 1 \\ x_n \end{pmatrix}$

$$\mathbf{Xw} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ & \vdots \\ 1 & x_N \end{bmatrix} \times \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$



기저함수Basis Functions

- 어떤 함수를 만들어 내기 위한 재료 함수
  - 2차식을 위한 재료 함수 : $1, x, x^2$
  - $3x^2 + 4x + 2 = 3 \times x^2 + 4 \times x + 2 \times 1$



$$f(x) = c_1 x^0 + c_2 x^1 + c_3 \sin(x) + c_4 \exp(x)$$

$$f(x) = c_1 x^0 + c_2 x^1 + c_3 x^3 + c_4 x^4$$

$$f(x) = c_1 \phi_1(x) + c_2 \phi_2(x) + c_3 \phi_3(x) + c_4 \phi_4(x)$$

임의의 함수로 표현

어떤 재료함수들의 조합이 아닐까?

다항식 기저Polynomial Basis

- 다항식으로 적당한 곡선을 만들자

$$y(x, \mathbf{w}) = w_0 x^0 + w_1 x^1 + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- 우리가 가진 데이터의 모양은 $y(x, \mathbf{w})$와 같지 않을까? → 가설
- 결정해야 할 것은?
  - 항은 몇 개까지 만들까? M=?
  - M+1개의 w들은 어떻게 설정할까?

다항 특성Polynomial Features

- 주어진 데이터 x 이외에 x^0, x^2, x^M이 추가 되었으므로 특성feature이 추가

$$y(x, \mathbf{w}) = w_0 x^0 + w_1 x^1 + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- 주어진 데이터가 주택 가격 관련 데이터라면.....
  - x : 주택의 너비, x^2 : 주택의 면적, x^3 : 주택 외관 전체의 부피 → 대상의 특성
  - t : 주택 가격

기저함수의 벡터 표현

- 아래 가설 $y(x, \mathbf{w})$을 벡터형식으로 표현

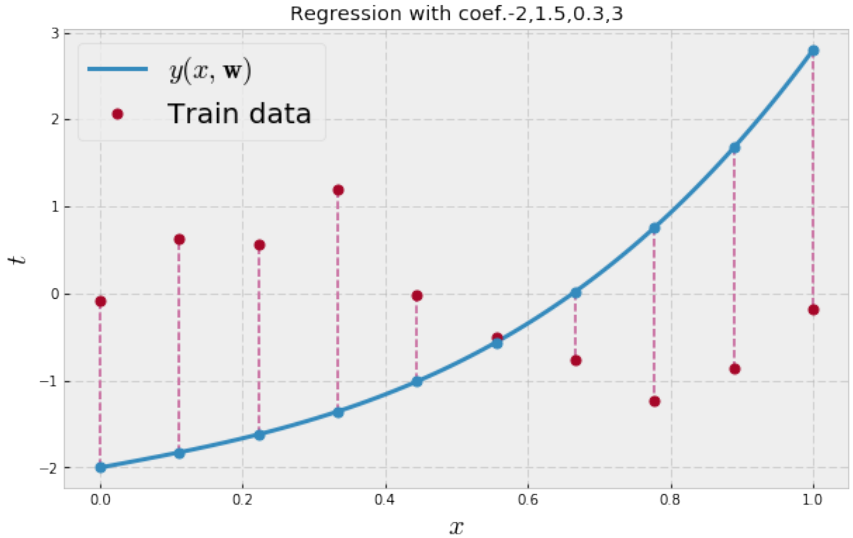
$$y(x, \mathbf{w}) = w_0 x^0 + w_1 x^1 + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$y(x, \mathbf{w}) = [w_0 \quad w_1 \quad w_2 \quad \cdots \quad w_M] \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ \vdots \\ x^M \end{bmatrix} = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(x) \qquad \boldsymbol{\phi}(x): \mathbb{R} \to \mathbb{R}^{M+1}$$

그냥 맘대로 적당히…

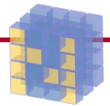
- 그냥 맘대로 적당히…
- $\mathrm{M} = 3, w_0 = -2, w_1 = 1.5, w_2 = 0.3, w_3 = 3$

$$y(x, \mathbf{w}) = -2 + 1.5x^1 + 0.3x^2 + 3x^3$$



numpy vs sklearn

- numpy, sklearn을 이용한 선형회귀
- 두 패키지에서 모두 회귀 알고리즘 제공



NumPy

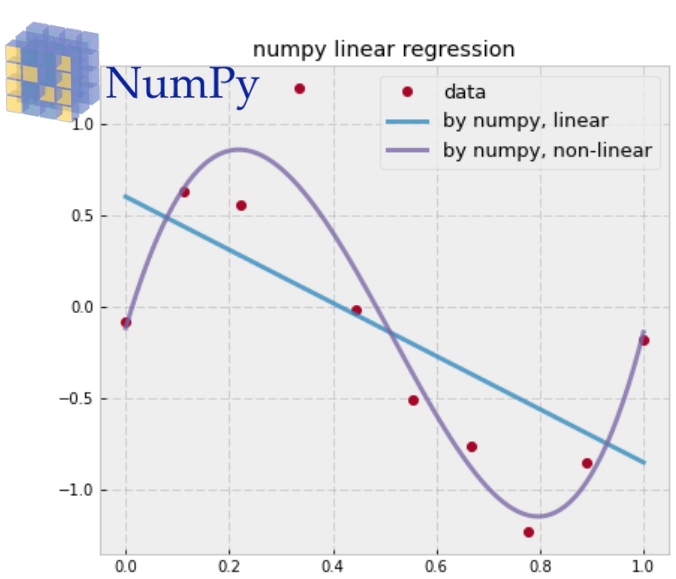
```
# 1차 다항식
z_lin = np.polyfit(x_train, t_train, 1)

# M차 다항식
z_nonlin = np.polyfit(x_train, t_train,
M)
```

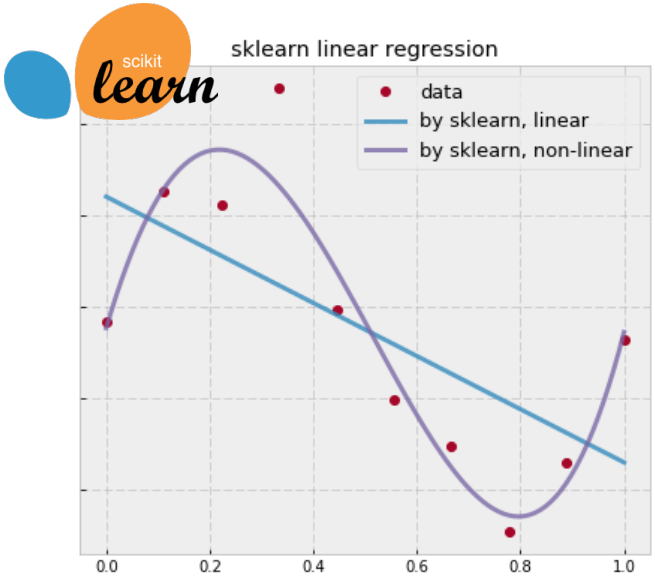
scikit learn

```
model =
LinearRegression(fit_intercept=True)
model.fit(x_train[:, np.newaxis],
t_train)

poly_model =
make_pipeline(PolynomialFeatures(M),
LinearRegression())
poly_model.fit(x_train[:, np.newaxis],
t_train)
```

numpy vs sklearn

- `numpy coef.   : [ 3.2748 14.1305 -27.1019 9.6745], Bias:-0.1181`
- `sklearn coef. : [ 3.2748 14.1305 -27.1019 9.6745], Bias:-0.1181`





직접 만들자!

- 이제 배운 지식을 무기로 numpy, sklearn에서 제공하는 알고리즘을 직접 만들어보자!
  - 경사하강법
  - 정규방정식 풀기

- 관련된 일반 이슈
  - 레귤러라이제이션 : L1, L2

경사하강을 이용한 선형회귀

- 목적 : 정의된 에러함수를 최소화
- 결과 : 에러를 최소로 하는 $\mathbf{w}$
- 도구 : 4주차에 만든 minimize 함수
- 추가사항
  - 과적합의 의미
  - 과적합을 방지하기 위한 규제항 추가

얼마나 못했나?measure P

- Error 함수, 로스함수, 코스트함수 : 목적함수

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 \quad (1.1) \qquad E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 \quad (1.1*)$$

  - $\mathbf{w}$: 결정해야 하는 회귀계수(벡터 변수)
  - x_n : 데이터의 입력 값(일반적으로 벡터 변수, 여기서는 스칼라)
  - t_n : 데이터의 타겟 값(여기서는 y 좌표값)

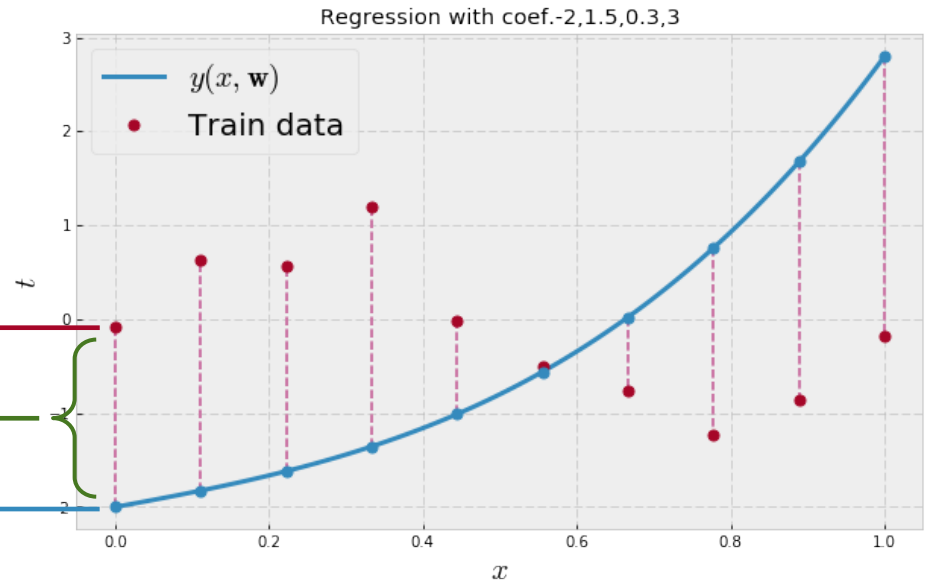
- 오차의 제곱을 최소화시키기를 원함 → 최소제곱법least square

에러함수Error Function

- 에러 함수

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$



에러값 계산

- 에러 함수

```
y = -2 + 1.5*x_train + 0.3*x_train**2 + 3*x_train**3
```

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

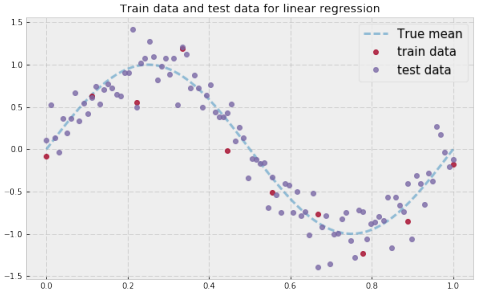
```
print(0.5*(((y - t_train)**2).sum()))
```

결과 : 20.917214052454877

- 이 값이 가능한 작아지기를 원한다.

가설과 에러 함수

데이터



가설(다항함수형태)

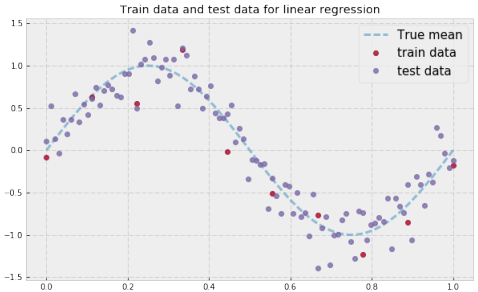
$$y(x, \mathbf{w}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(x)$$

에러함수(목적함수)

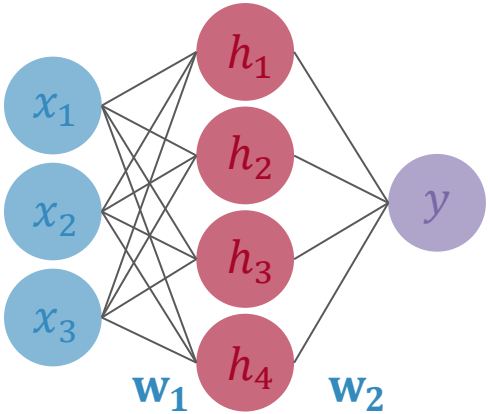
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

가설과 에러 함수

데이터



가설(신경망 형태)



$= y(x, \mathbf{w})$

에러함수(목적함수)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

에러함수 코드Error Function Code

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

```
def E(w, M, x, t):
    """
    E(w)= (1/2) * sum_{n=1}^{N} {y(x_n,w) - t_n}^2
    y(x_n, w) = w_0*x^0 + w_1*x^1 + w_2*x^2 + ... + w_M*x^M
    """
    X = np.array([ x**i for i in range(M+1) ])
    y = (w.reshape(-1,1) * X).sum(axis=0)

    return 0.5*(( (y - t)**2 ).sum())
```

에러함수의 경사도벡터

- 수치 미분 함수를 사용
- 또는 직접 미분

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left(\sum_{n=1}^{N} (\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) - t_n) \right) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

$$= \mathbf{w}^{\mathrm{T}} \left(\sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}_n) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} \right) - \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

$$= \mathbf{w}^{\mathrm{T}} \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right) - \mathbf{t}^{\mathrm{T}} \boldsymbol{\Phi}$$

```
def grad_anal(w, M, x, t):
    """
    This function computes the analytic gradient of the objective function
    x, t : data for error function eval.
    """
    N = x.shape[0]
    PI = np.hstack( (np.power(x.reshape(N,1), p) for p in range(M+1)) )
    g = np.dot(w.T, np.dot(PI.T, PI) ) - np.dot(t.T, PI)

    return g.astype('longdouble')
```

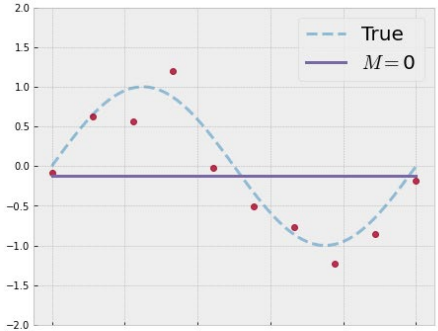
auto_diff_test.ipynb

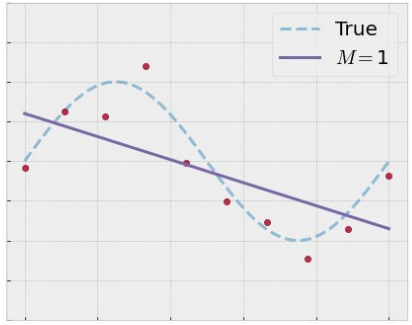
경사하강 실험 코드

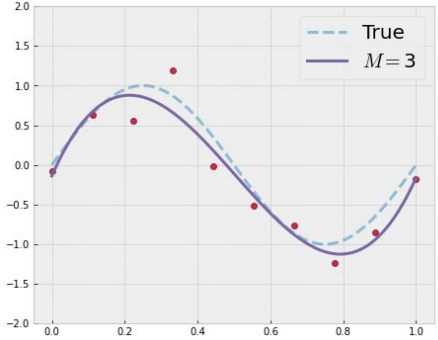
- 정의된 에러함수(목적함수)와 목적함수의 경사도벡터 함수를 이용하여 경사하강 시도

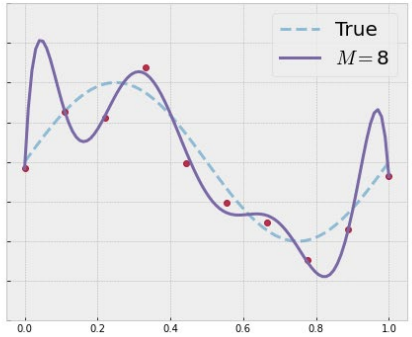

```
# M을 0, 1, 3, 9에 대해서 한번씩 피팅
ms = [0, 1, 3, 8]

# 피팅결과 구해진 파라미터를 W에 저장
W = []

# 학습데이터를 넘기면서 fitting
for M in ms :
    x0 = np.random.uniform(-1, 1, M+1)
    W.append(minimize(E, x0, args=(M, x_train, t_train),
                      jac=grad_anal, method="CGFR", verbose_step=500))
```

경사하강 실험 결과









	M=0	M=1	M=3	M=8
0	-0.12627	0.599645	-0.139682	-0.082299
1	NaN	-1.451831	10.483091	90.751711
2	NaN	NaN	-31.185369	-1698.272058
3	NaN	NaN	20.680189	12647.995512
4	NaN	NaN	NaN	-47354.721824
5	NaN	NaN	NaN	97608.618543
6	NaN	NaN	NaN	-112611.383684
7	NaN	NaN	NaN	68191.475090
8	NaN	NaN	NaN	-16874.563844

에러 평가

- RMS(Root Mean Square)

$$E_{RMS} = \sqrt{\frac{2E(\mathbf{w}^*)}{N}}$$

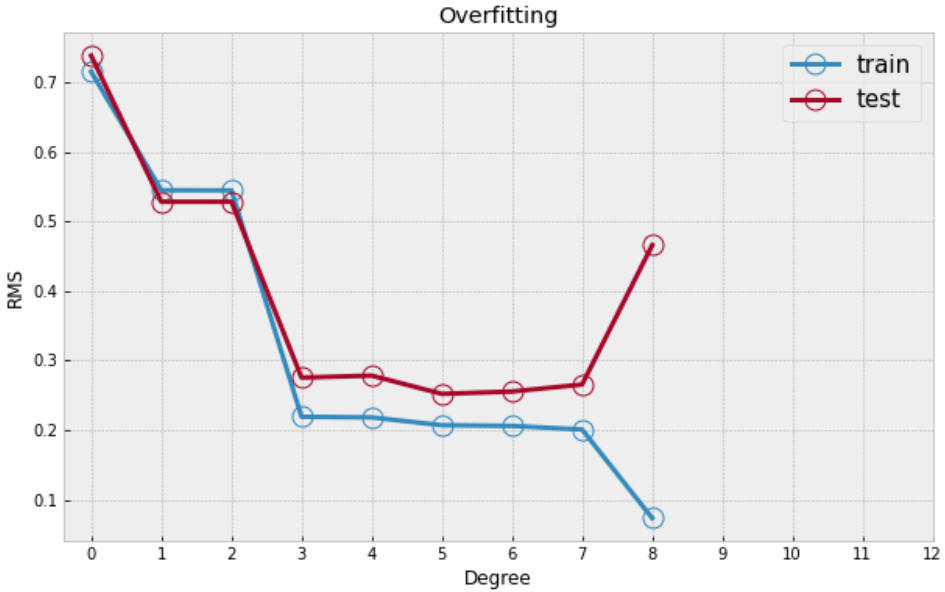
```
def rms(w, E, M, x, t) :
    return np.sqrt( (2*E(w, M, x, t))
                    / x.shape[0] )
```

- 루트 : 오차 함수값의 제곱을 상쇄
- N으로 나누어 평균적인 오차를 구함
- 2 곱하기는 오차함수에 (1/2) 때문
- 1차에서 8차까지 회귀 결과에 대해 학습, 테스트 데이터로 각각 RMS를 계산

과적합(오버피팅Overfitting)

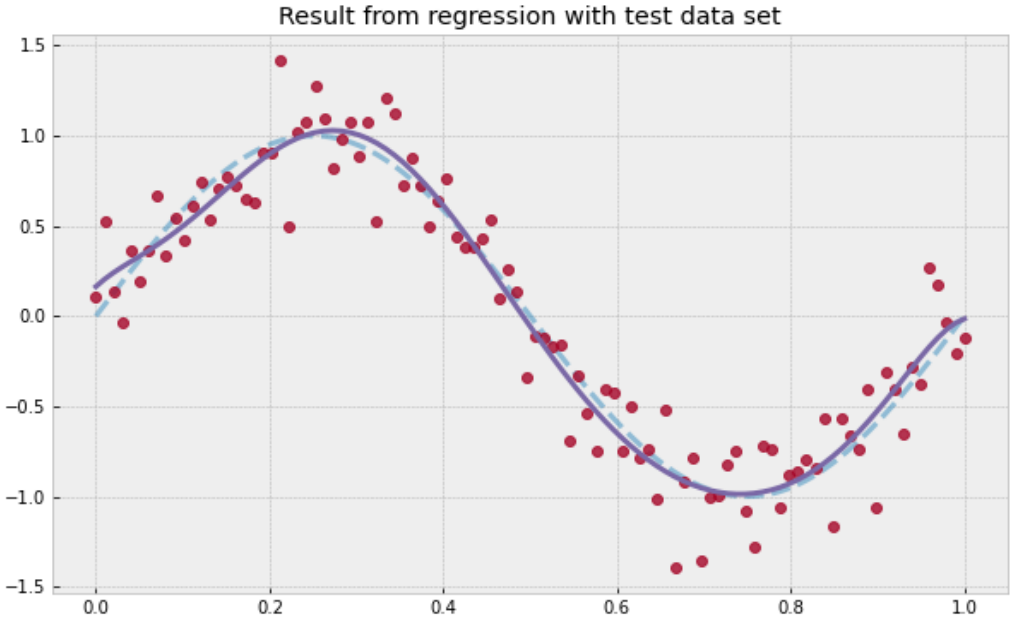
- 학습데이터에 대한 에러는 감소
- 테스트데이터에 대한 에러는 증가

- 해결
  - 더 많은 데이터
  - 레귤러라이제이션Regularization



과적합 해결 (1)

- 더 많은 데이터를 모아서……



과적합 해결 (2)

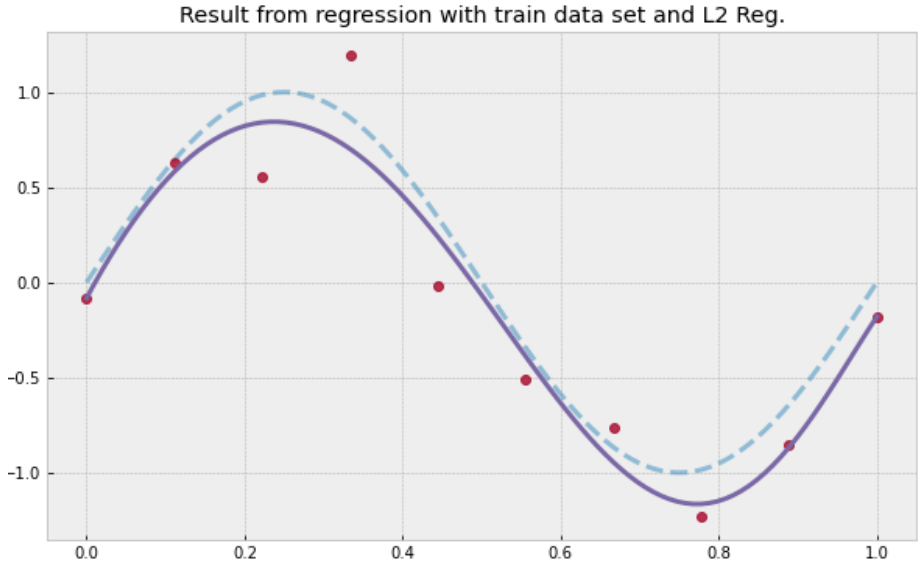
- 레귤러라이제이션Regularization
  - 일반화, 규제화, 정규화
  - 목적함수에 페널티 항을 더해서

$$\tilde{E} = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 \quad + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$\tilde{E}$가 커진다. $\mathbf{w}$가 커지면

과적합 해결 (2)

- L2 레귤러라이제이션
- W:[-0.0818 7.7064 -14.9829 -6.3789 5.8483 9.1046 5.3738 -0.6798 -6.0898]



제약최소화와 규제화†

- 페널티 항의 일반적 모양

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$

- 제약최소화 문제

$$\text{Minimize} \quad \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

$$s.t. \sum_{j=1}^{M}|w_j|^q \leq \eta$$

jupyter lin_reg.ipynb

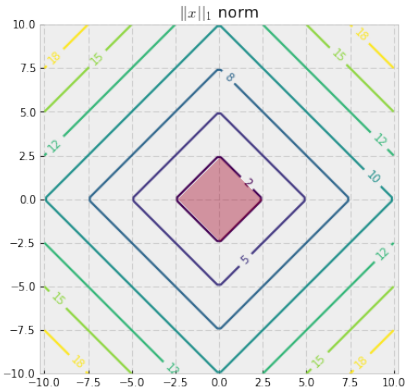
제약최소화와 규제화

- 규제항에 따라 w가
  등고선 영역 안에 갇힘(제약)

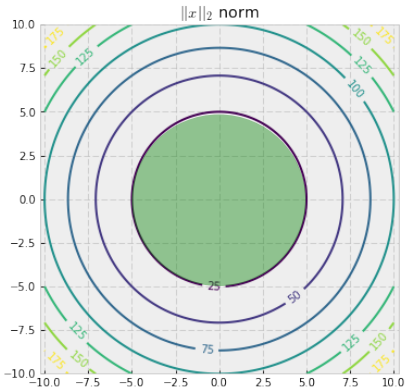
$$s.t. \sum_{j=1}^{M} |w_j|^q \leq \eta$$

$q = 1, \eta = 2$

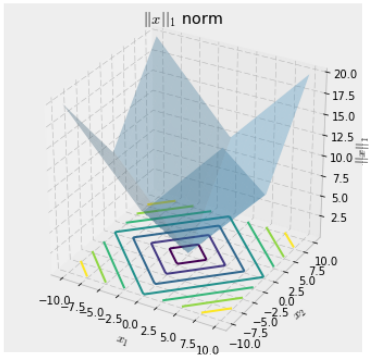
$q = 2, \eta = 25$

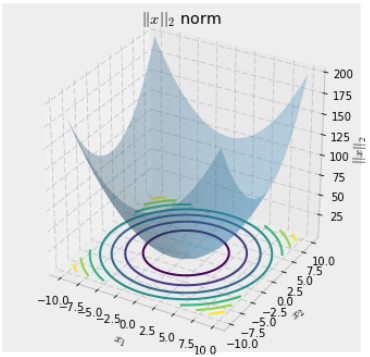


L1



L2





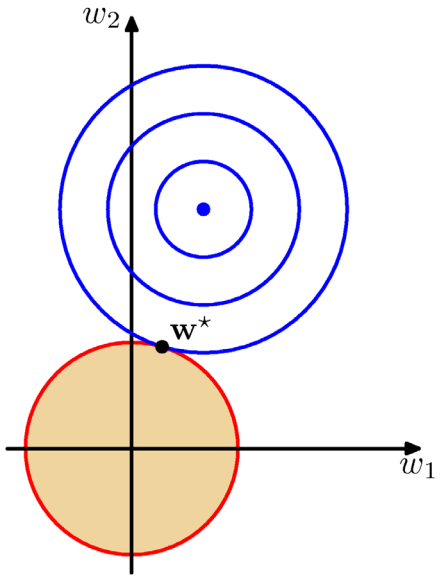
제약최소화와 규제화

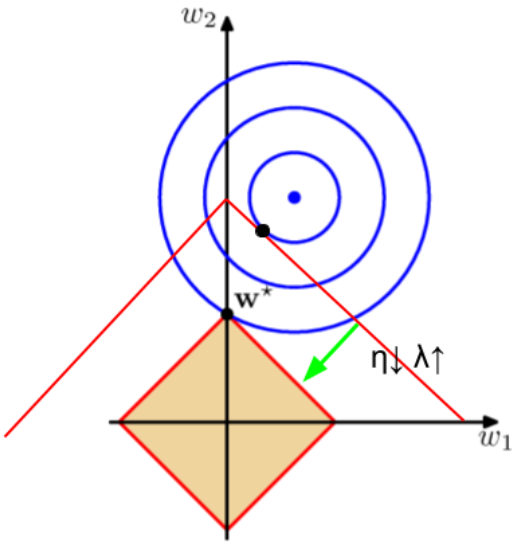
- λ 증가, η 감소: 레귤러라이제이션 효과 커짐

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$

$$\text{Minimize} \quad \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

$$s.t. \sum_{j=1}^{M}|w_j|^q \leq \eta$$





레귤러라이제이션 실험

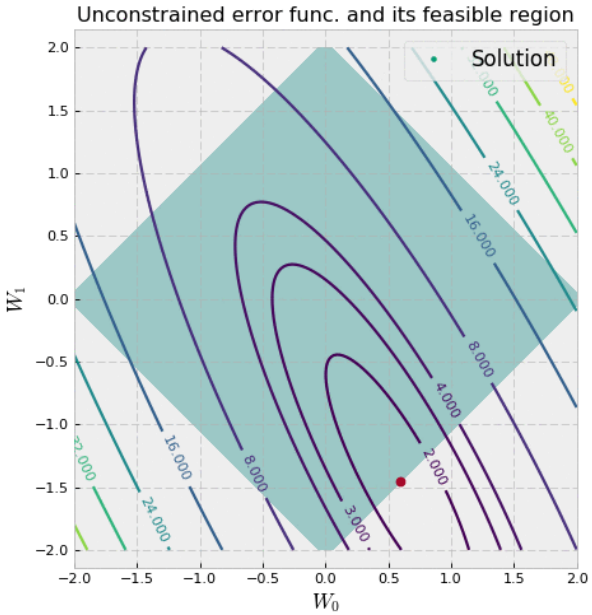
- 에러함수 수정 $\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$

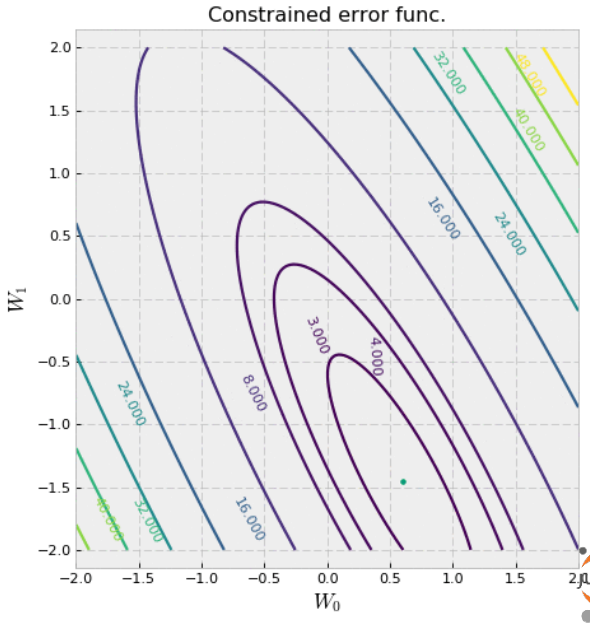
w_0를 규제항에 포함 할지 말지 결정

```
def E(w, M, x, t):
    ……
    if regularizer == 'L2' :
        # L2 reg
        if not reg_intercept :
            return 0.5*(( (y - t)**2 ).sum()) + (lamda/2.)*np.linalg.norm(w[1:])**2
        else :
            return 0.5*(( (y - t)**2 ).sum()) + (lamda/2.)*np.linalg.norm(w)**2
    elif regularizer == 'L1' :
        # L1 reg.
        if not reg_intercept :
            return 0.5*(( (y - t)**2 ).sum()) + (lamda/2.)*(np.abs(w[1:]).sum())
        else :
            return 0.5*(( (y - t)**2 ).sum()) + (lamda/2.)*(np.abs(w).sum())
    else :
        return 0.5*(( (y - t)**2 ).sum())
```

L1 Regularization

- $0 \leq \lambda \leq 3$

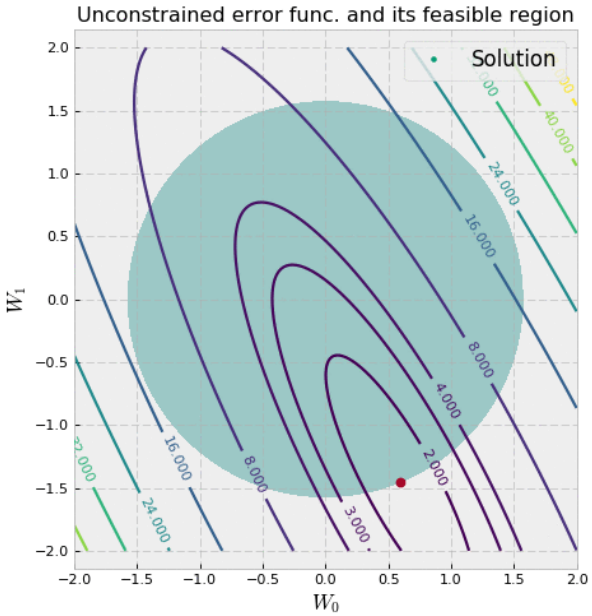


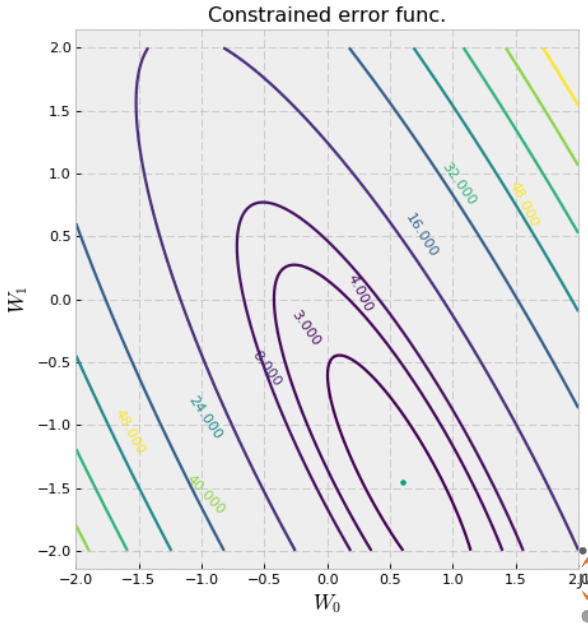


jupyter lin_reg.ipynb

L2 Regularization

- $0 \leq \lambda \leq 15$





jupyter lin_reg.ipynb

정규방정식을 이용한 선형회귀

- 목적 : 주어진 목적함수를 만족시키는 $\mathbf{w}$를 한번에 찾는다.
- 결과 : 에러를 최소로 하는 $\mathbf{w}$
- 도구 : 미분
- 추가사항†
  - 최소제곱의 기하학적 의미
  - L2 규제가 적용된 경우 정규방정식 해

가설의 표현

- $y(\mathbf{x}_n, \mathbf{w}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)$: D차원 벡터에 대한 기저함수 형식
  - $\mathbf{x}_n$: D차원 벡터인 입력 데이터
  - $\boldsymbol{\phi}(\mathbf{x}_n)$: $\mathbb{R}^D \rightarrow \mathbb{R}^M$ 인 다변수 벡터함수, $\phi_j(\mathbf{x})$: 다변수 실함수
  - M : 기저함수의 개수
  - $\mathbf{w} = (w_0 \quad \cdots \quad w_{M-1})^{\mathrm{T}}$, $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}) \quad \cdots \quad \phi_{M-1}(\mathbf{x}))^{\mathrm{T}}$
  - $D = 1$이고 다항 기저인 경우

$$\boldsymbol{\phi}(x) = \begin{Bmatrix} \phi_0(x) \equiv x^0 \\ \phi_1(x) \equiv x^1 \\ \vdots \\ \phi_{M-1}(x) \equiv x^{M-1} \end{Bmatrix}$$

- $y(\mathbf{x}_n, \mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_n$: D차원 벡터를 그대로 사용
  - $\mathbf{w} = (w_0 \quad \cdots \quad w_{D-1})^{\mathrm{T}}$, $\mathbf{x} = (x_0 \quad \cdots \quad x_{D-1})^{\mathrm{T}}$

목적함수

- 기저함수를 쓰는 경우
  - $E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$
  - N : 데이터의 개수
  - $\mathbf{w}$ 의 차원 : 기저함수의 개수 M

$$[w_0 \quad w_1 \quad \cdots \quad w_{M-1}] \begin{bmatrix} x_n^0 \\ x_n^1 \\ \vdots \\ x_n^{M-1} \end{bmatrix}$$

- 입력변수에 대해 선형인 경우
  - $E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\mathbf{x}_n\}^2$
  - N : 데이터의 개수
  - $\mathbf{w}$ 의 차원 : 입력변수와 동일한 D

$$[w_0 \quad w_1] \begin{bmatrix} 1 \\ x \end{bmatrix}$$

스칼라를 벡터로 미분

- 분모 레이아웃

$$\frac{\partial \mathbf{x}^{\mathrm{T}}\mathbf{a}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial x_i a_i}{\partial x_1} \\ \frac{\partial x_i a_i}{\partial x_2} \\ \vdots \\ \frac{\partial x_i a_i}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \mathbf{a}$$

- 분자 레이아웃

$$\frac{\partial \mathbf{x}^{\mathrm{T}}\mathbf{a}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial x_i a_i}{\partial x_1} & \frac{\partial x_i a_i}{\partial x_2} & \cdots & \frac{\partial x_i a_i}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} = \mathbf{a}^{\mathrm{T}}$$

목적함수의 미분

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

$$\frac{\partial E_D(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left\{\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2\right\}$$

$$\frac{\partial}{\partial \mathbf{w}} + \frac{\partial}{\partial \mathbf{w}} + \ldots + \frac{\partial}{\partial \mathbf{w}}$$

$$= \frac{1}{2}\sum_{n=1}^{N} 2\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}\left(\frac{\partial}{\partial \mathbf{w}}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}\right)$$

$$= \sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}\left(-\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right) = -\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = 0$$

목적함수의 미분

$$\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = 0$$

$$\sum_{n=1}^{N}\{t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\} = 0$$

$$\sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} - \mathbf{w}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right) = 0$$

$$\mathbf{w}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right) = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

정규방정식

$$\mathbf{w}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right)=\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

$$\mathbf{w}^{\mathrm{T}}\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)=\mathbf{t}^{\mathrm{T}}\boldsymbol{\Phi}$$

$$\left\{\mathbf{w}^{\mathrm{T}}\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)\right\}^{\mathrm{T}}=\left\{\mathbf{t}^{\mathrm{T}}\boldsymbol{\Phi}\right\}^{\mathrm{T}}$$

$$\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)\mathbf{w}=\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

$$\mathbf{w}=\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

Moore-Penrose
Pseudo Inverse

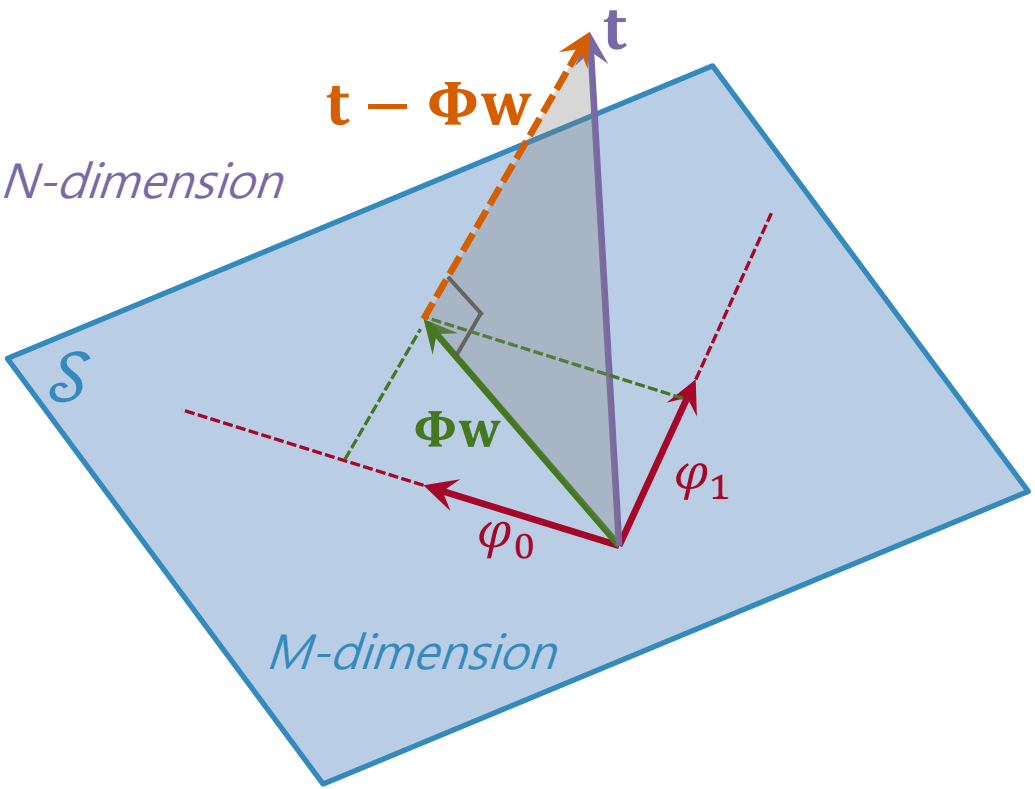
$$\underset{\boldsymbol{\Phi}}{\begin{bmatrix}\phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1)\\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2)\\ \vdots & \vdots & \ddots & \vdots\\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N)\end{bmatrix}}\underset{\mathbf{w}}{\begin{bmatrix}w_0\\ w_1\\ \vdots\\ w_{M-1}\end{bmatrix}}\overset{?}{=}\underset{\mathbf{t}}{\begin{bmatrix}t_1\\ t_2\\ \vdots\\ t_N\end{bmatrix}}$$

Design Matrix

$$\mathbf{w}=\boldsymbol{\Phi}^{-1}\mathbf{t}$$

열(col.)들의 선형조합으로 **t** 를 만들 수 없다.

최소제곱의 기하학적 의미



정규방정식 실험

- Step 1. 주어진 데이터 $\mathcal{D} = (\mathbf{t}, \mathbf{X})$, $\mathbf{t} = (t_1, t_2, \dots, t_N)^{\mathrm{T}}$, $\mathbf{X} = \left(\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}, \dots, \mathbf{x}_N^{\mathrm{T}}\right)^{\mathrm{T}}$
- Step 2. 필요하다면 기저함수를 M개 선택한다.
- Step 3. 디자인 행렬을 구성한다.

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_N^{\mathrm{T}} \end{bmatrix}$$

- Step 4. 정규방정식을 푼다.

$$\mathbf{w} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \qquad \mathbf{w} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t}$$

정규방정식 실험 코드

- 디자인 행렬 구성

$\mathbf{X} =$

```
[[0.      1. ]
 [0.1111  1. ]
 [0.2222  1. ]
 [0.3333  1. ]
 [0.4444  1. ]
 [0.5556  1. ]
 [0.6667  1. ]
 [0.7778  1. ]
 [0.8889  1. ]
 [1.      1. ]]
```

$\mathbf{\Phi} =$

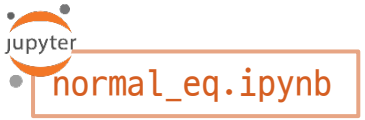
x^0	x^1	x^2	x^3
1.	0.	0.	0.
1.	0.1111	0.0123	0.0014
1.	0.2222	0.0494	0.011
1.	0.3333	0.1111	0.037
1.	0.4444	0.1975	0.0878
1.	0.5556	0.3086	0.1715
1.	0.6667	0.4444	0.2963
1.	0.7778	0.6049	0.4705
1.	0.8889	0.7901	0.7023
1.	1.	1.	1.

```
X  = np.append( x_train.reshape(-1, 1), np.ones(N).reshape(-1,1), axis=1 )
PI = np.hstack( (np.power(x_train.reshape(-1,1), p) for p in range(M)) )
```

정규방정식 실험 코드

- 정규방정식 풀이

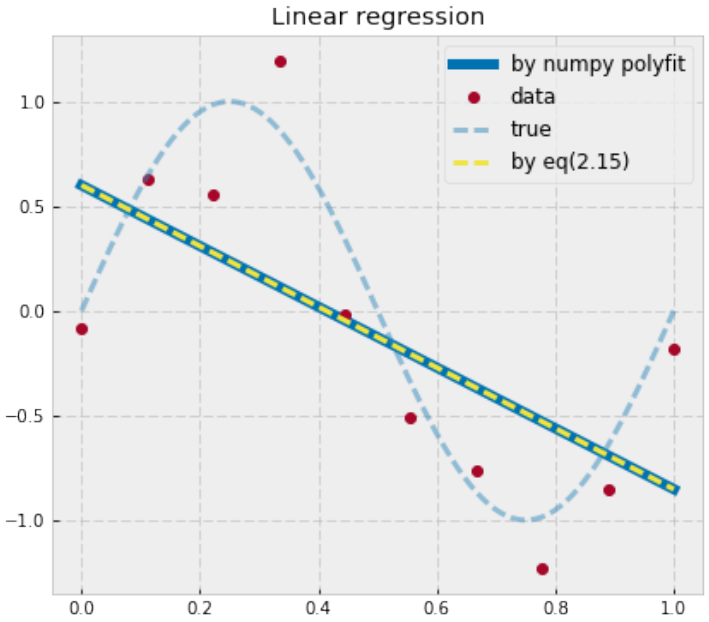
$$\mathbf{w} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t} \qquad \mathbf{w} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

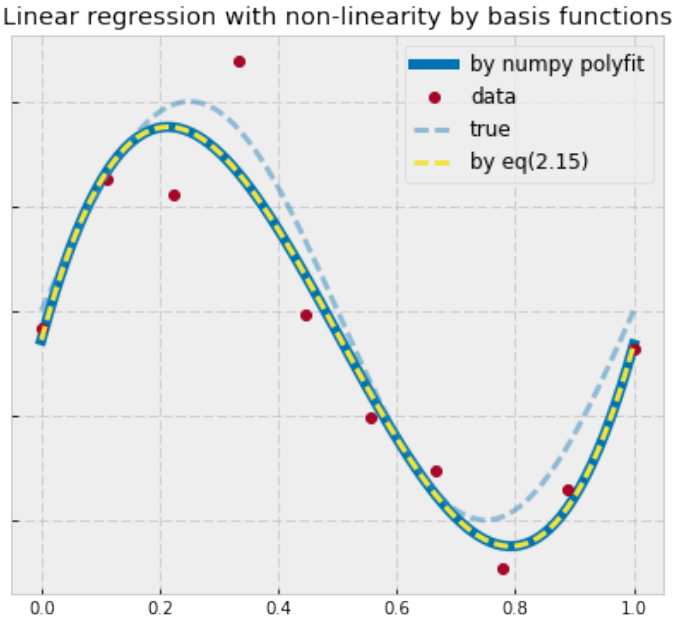



```
w_lin = np.dot( inv(np.dot(X.T, X)) ,
                np.dot(X.T , t_train.reshape(N,1)) ).reshape(-1)

w_lin = np.linalg.solve(np.dot(X.T, X),
                        np.dot(X.T, t_train.reshape(N,1)) ).reshape(-1)

w_nonlin = np.dot( inv(np.dot(PI.T, PI)) ,
                   np.dot(PI.T, t_train.reshape(N,1)) ).reshape(-1)[::-1]

w_nonlin = np.linalg.solve(np.dot(PI.T, PI),
                           np.dot(PI.T, t_train.reshape(N,1)) ).reshape(-1)[::-1]
```

정규방정식 실험 결과

- numpy polyfit 함수와 결과 비교



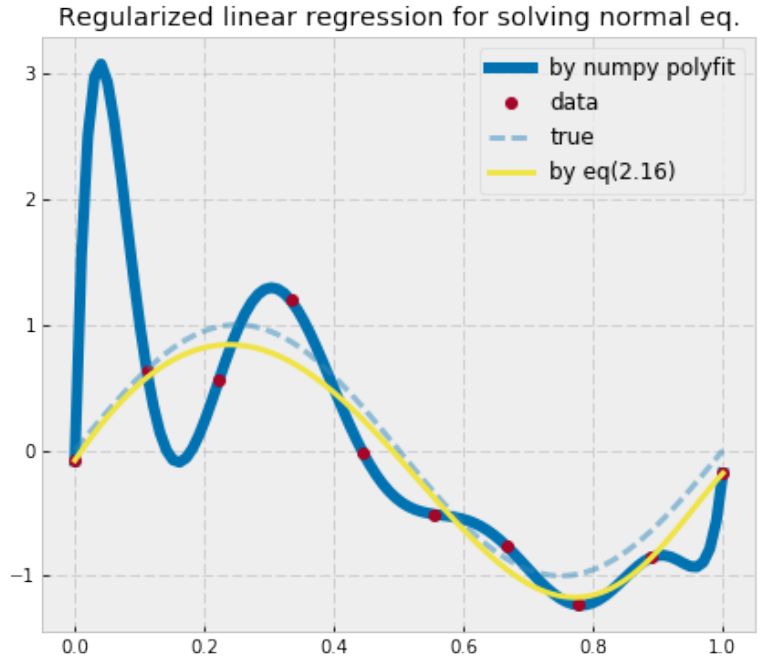


정규방정식의 규제항

- L2 레귤러라이제이션을 적용한 정규방정식

$$\mathbf{w} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

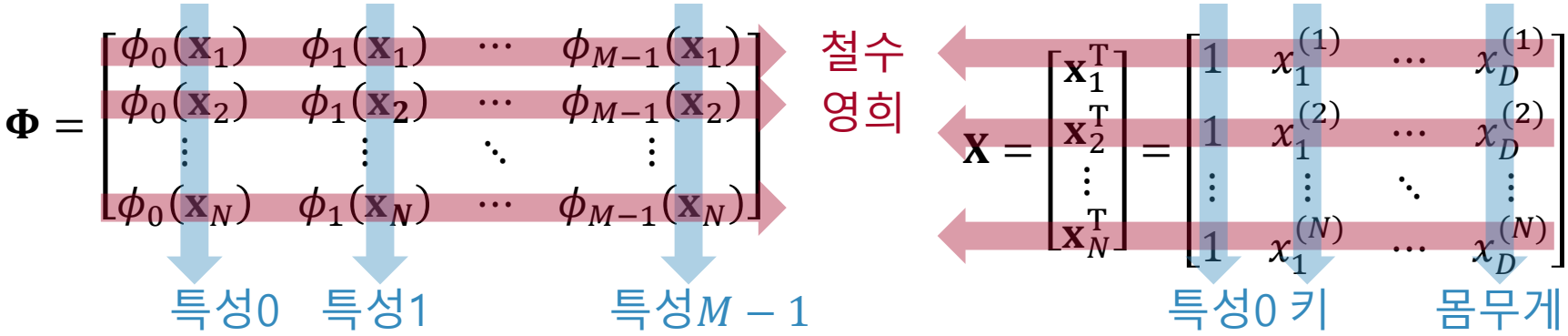
$$\mathbf{w} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda\mathbf{I}_M\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$



jupyter nomal_eq.ipynb

다변수 선형회귀

- 입력변수의 차원만 증가 $D = 1 \rightarrow D = 2,3,4, \ldots$ 데이터 → 특성



다변수 스칼라 함수

$\mathbf{x}_1 \rightarrow \phi_j(\quad) \rightarrow f_j^{(1)}$

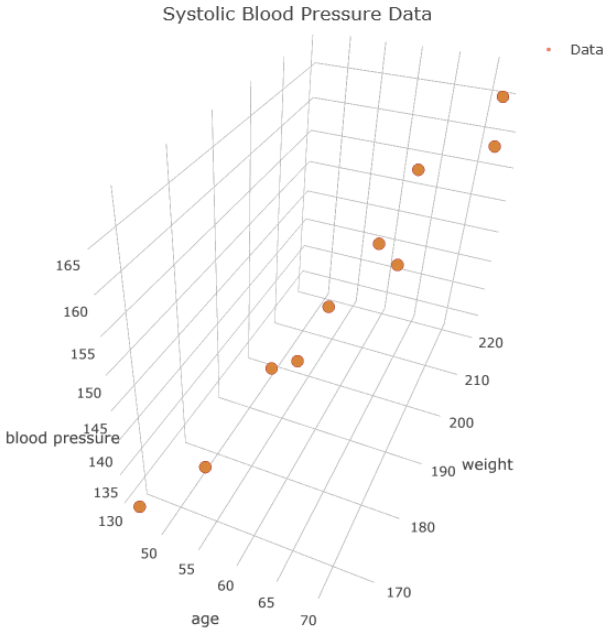
숫자 D개인 벡터가 들어가서

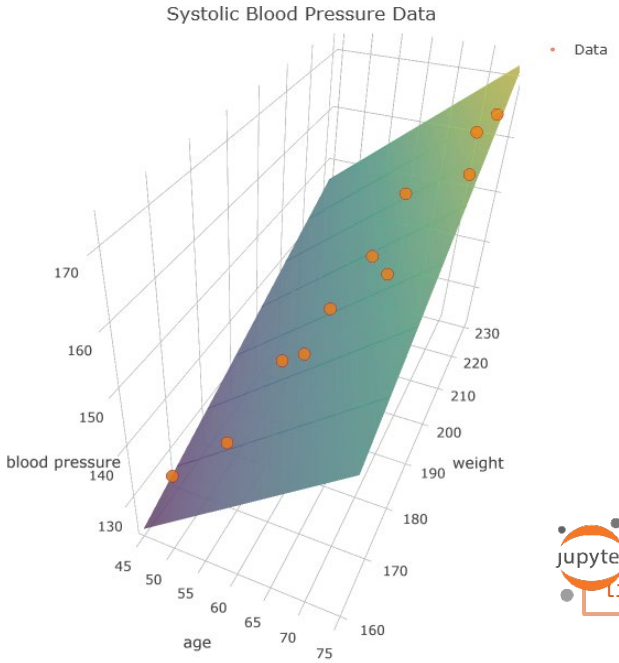
1번 데이터에 대한 j번째 특징 출력

$x_j^{(i)}$ i번 데이터에 대한 j번째 특징

다변수 선형회귀

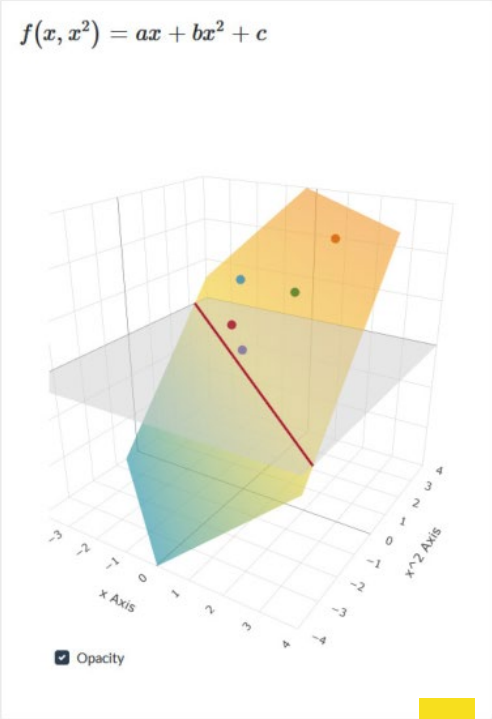
- 입력변수의 차원만 증가 $D = 2, \mathbf{X} = (11,3)$

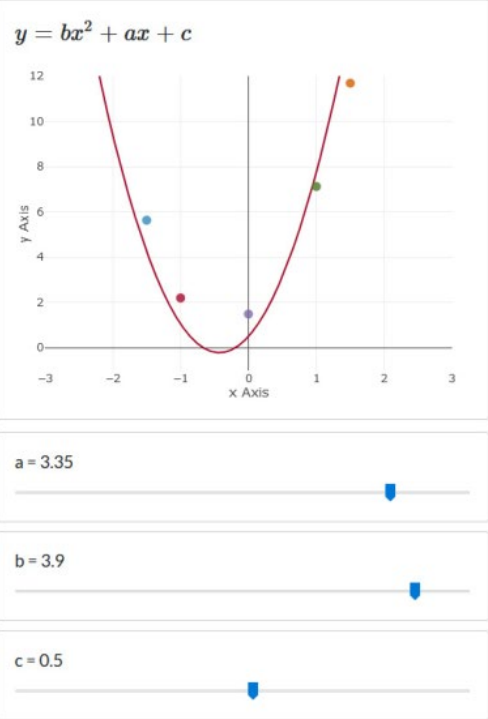




jupyter lin_reg_multi_fit.ipynb

선형회귀의 선형성





JS metamath1.github.io/noviceml/linearity.html