Regression, Gradient Descent

Machine Learning

(PM chap 2, 10)

Regression

- From x → predict y

x

y

Size of a house is 1800 ft^2

price ?

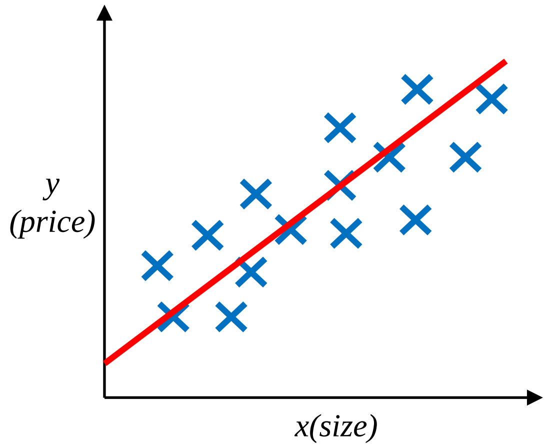
Regression

- Predicting value y
  - From x (input value) → predict y (output/target value, continuous)
  - *Regression*: constructing a model ***y = f(x)*** , using training data (x, y)

target value

x size(ft^2)		y price($)
2104		400,000
1600		330,000
2400		369,000
3000		540,000
…	…	…

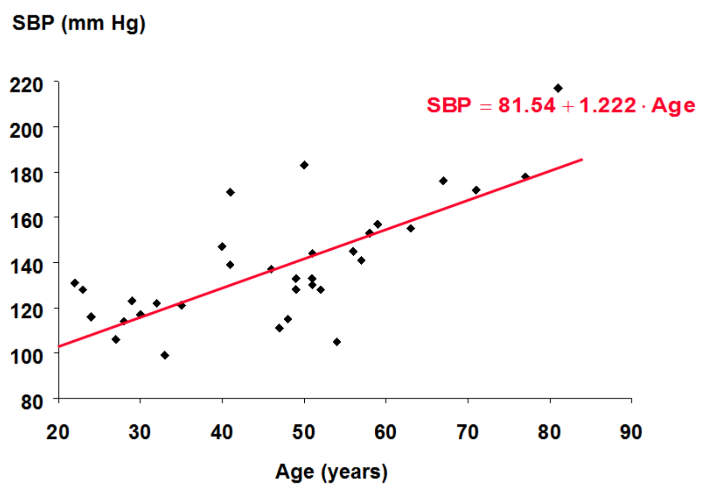
1800 → price ?

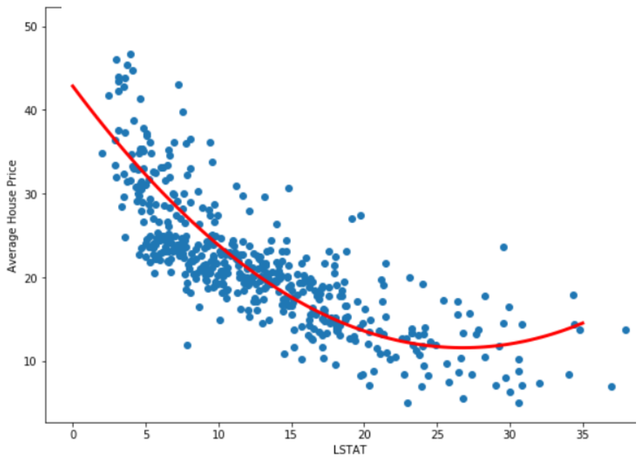


input x (area) → $\boxed{f(x)}$ → output y (price)

Regression

- Examples
  - Age vs. Blood pressure
  - Lower status % vs. House price





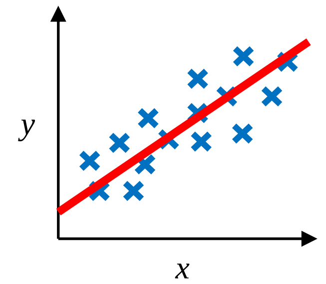
dongguk UNIVERSITY

Linear Regression

- The model
  - y = f(x) is modeled as a linear function

$$y = f_\theta(x) = w \cdot x + b$$

parameters
(*w*, *b*)



x	y
2104	400
1600	330
2400	369
3000	540
…	…

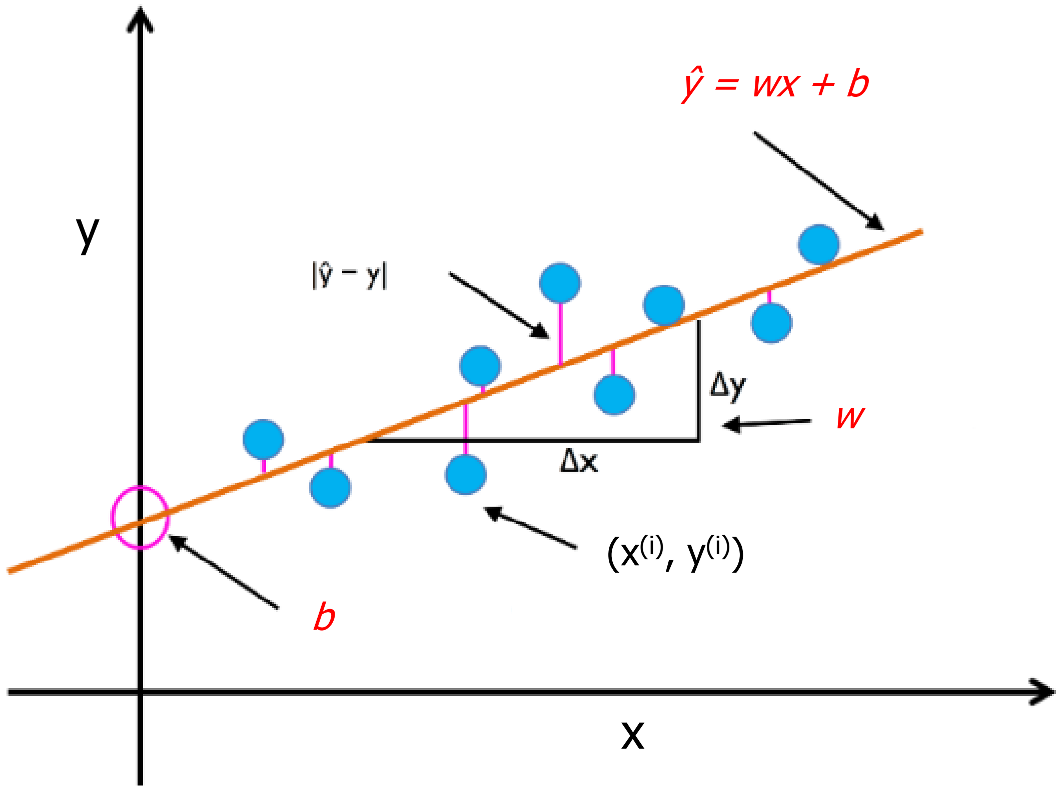
  - *w* and *b* are estimated using the training data $(x^{(i)}, y^{(i)})$
    - *w* : weight (coefficient)
    - *b* : bias (intercept)

    θ : parameters

  - Minimize prediction error → find *w* and *b*
    - Least square method
    - Gradient descent

Estimating Parameters

- Finding optimal *w*, *b*
  - Minimize the difference between actual value ***y*** and predicted value ***ŷ***



Cost Function

- Cost(loss) function
  - A measure of how wrong the model is in terms of its ability to estimate the relationship between x and y
  - Goal of learning is to ***minimize the cost function***
- Data

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$$

- Prediction model

$$\hat{y} = f_\theta(x)$$

- MSE(mean square error) cost function

$$\boxed{J(\boldsymbol{\theta}) = \frac{1}{2m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)^2 \quad = \frac{1}{2m}\sum\left(f_\theta(x^{(i)}) - y^{(i)}\right)^2}$$

Gradient Descent

- Gradient descent
  - An iterative method for finding parameters to minimize a function

    Goal: $\underset{\theta}{\text{minimize}}\, J(\theta)$

- Gradient
  - Gradient of a multivariate function

    = vector of derivatives (slopes)

$$\nabla J(\theta) = \left(\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_m}\right)$$

    = direction of fastest increase

Gradient Descent

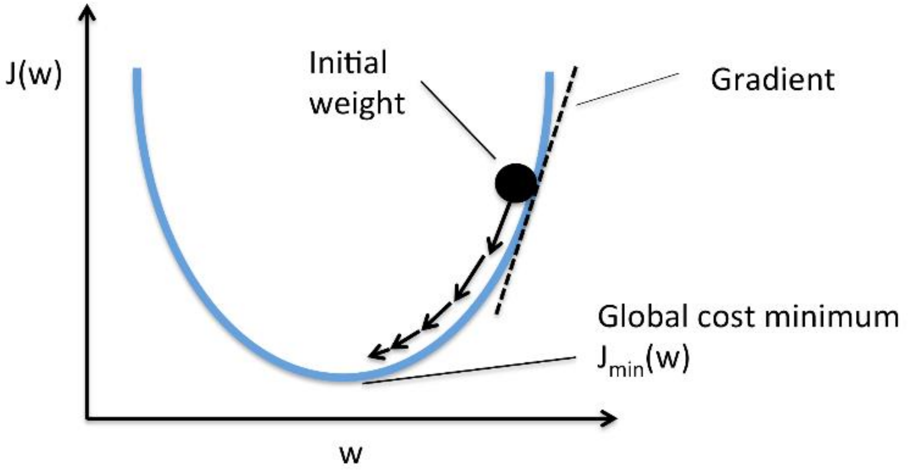
- Takes steps proportional to the *negative of the gradient*
- If the model has 1 parameter *w*,
  - Repeat

    If $\frac{\partial J}{\partial w} > 0$ → decrease w

    $\frac{\partial J}{\partial w} < 0$ → increase w

$$\therefore \quad \boxed{w = w - \alpha \frac{\partial J}{\partial w}}$$

α: learning rate

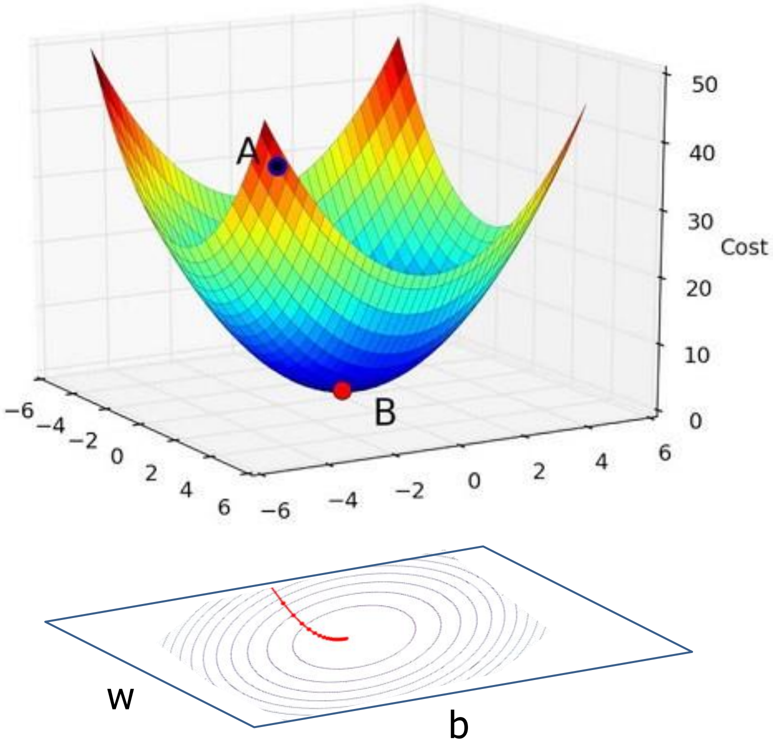


Gradient Descent

- If the model has parameters *w* and *b*,
  - Repeat

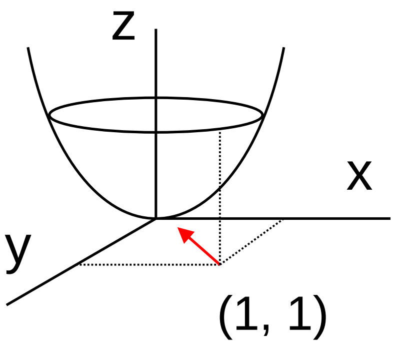
$$w = w - \alpha \frac{\partial J}{\partial w}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$



Gradient Descent

- Example
  - Function $z = f(x, y) = x^2 + y^2$



  - Gradient $\left(\frac{\partial f}{\partial x} = 2x, \quad \frac{\partial f}{\partial y} = 2y\right)$
  - At (x=1, y=1) → gradient is $(2,\ 2)$

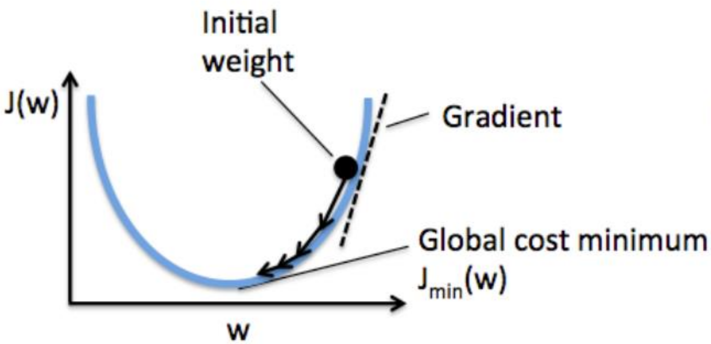
    ∴ The direction that most rapidly reduce z is (-2, -2)
  - For α = 0.1,

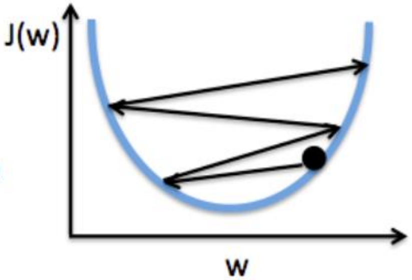
$$x = x - \alpha \cdot \frac{\partial f}{\partial x} = 1 - 0.1 \cdot 2 = 0.8$$

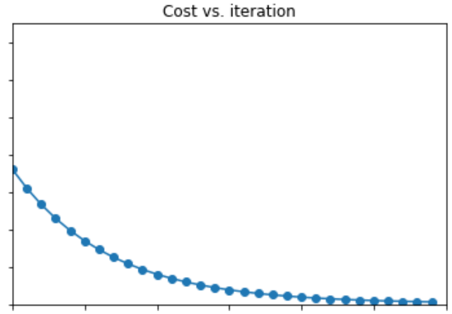
$$y = y - \alpha \cdot \frac{\partial f}{\partial y} = 1 - 0.1 \cdot 2 = 0.8$$

Learning Rate

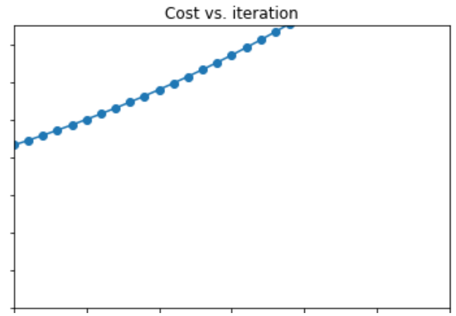
- Effect of learning rate α







small α



large α

GD for Linear Regression

- Data $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$
- Model $\hat{y} = wx + b$
- Cost $J(w, b) = \dfrac{1}{2m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)^2$
- Gradients

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w} \frac{1}{2m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) \frac{\partial}{\partial w} \left(wx^{(i)} + b\right)$$

$$= \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)$$

GD for Linear Regression

- Learning (gradient descent)

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

Initialize w, b

Repeat

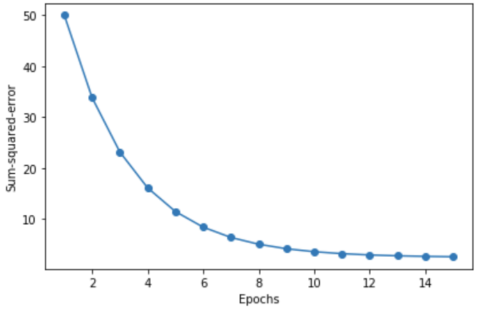
$$\hat{y}^{(i)} = wx^{(i)} + b$$

$$w = w - \alpha \frac{\partial J}{\partial w}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

$$\boxed{\begin{aligned} w &= w - \alpha \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) x^{(i)} \\ b &= b - \alpha \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) \end{aligned}}$$

$$cost = \frac{1}{2m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$



Example

- Data

```
x_train = np.array([1.8, 1.3, 1.7, 1.9, 1.4])
y_train = np.array([430., 320., 390., 490., 400.])
```

- Compute $\hat{y}$ and cost

```
    for i in range(m):
        y_hat = w * x[i] + b

        cost += (y_hat - y[i]) ** 2

    cost = (1 / (2 * m)) * cost
```

$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- Compute gradient

```
    for i in range(m):
        y_hat = w * x[i] + b

        dj_dw += -(y[i] - y_hat) * x[i]
        dj_db += -(y[i] - y_hat)

    dj_dw = dj_dw / m
    dj_db = dj_db / m
```

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)$$

Example

- Learning (gradient descent)

```python
def gradient_descent(x, y, w, b, alpha, num_iters):

    J_history = []
    for i in range(num_iters):
        dj_dw, dj_db = compute_gradient(x, y, w, b)

        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        J_history.append(compute_cost(x, y, w, b))
```

$$w = w - \alpha \frac{\partial J}{\partial w}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

```python
w_init = 0
b_init = 0
alpha = 0.01
iterations = 10000

w_final, b_final, J_hist = gradient_descent(x_train ,y_train, w_init, b_init, alpha, iterations)
```

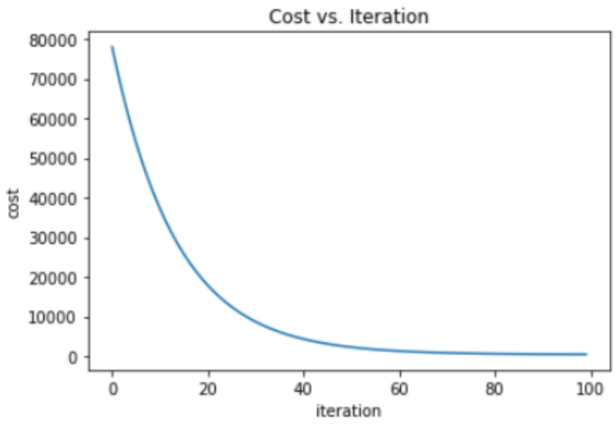
```
Iteration    0: Cost 7.79e+04
Iteration 1000: Cost 3.99e+02
Iteration 2000: Cost 3.96e+02
Iteration 3000: Cost 3.93e+02
Iteration 4000: Cost 3.92e+02
Iteration 5000: Cost 3.90e+02
```

```
w = 201.1031304050341
b = 80.34795484298294
```

Example

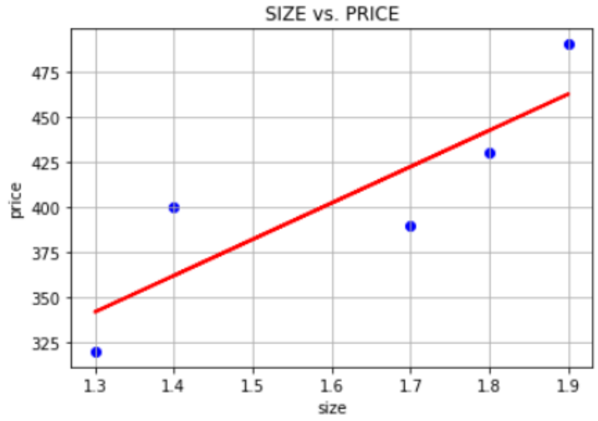
- Cost change

```
plt.plot(J_hist[:100])
```



- Learned model

```
y_hat = predict(x_train, w_final, b_final)
plt.plot(x_train, y_hat, color='red')
```

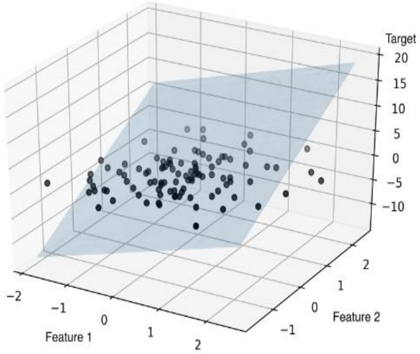


Multiple Linear Regression

- The model
  - y = f(x) is modeled as a linear function

$$y = f_{\theta}(x) = w_0 \cdot x_0 + w_1 \cdot x_1 + b$$

parameters
(w_0 , w_1 , b)



x0	x1	y
2104	4	400
1600	2	330
2400	3	369
3000	5	540
…		…

  - w_0, w_1 and b are estimated using the training data $(x^{(i)}, y^{(i)})$
    - w_0, w_1 : weight (coefficient)
    - b : bias (intercept)

θ : parameters

GD for Multiple Linear Regression

- Data $\left(x_0{}^{(1)}, x_1{}^{(1)}, \dots, y^{(1)}\right), \left(x_0{}^{(2)}, x_1{}^{(2)}, \dots, y^{(2)}\right), \dots, \left(x_0{}^{(m)}, x_1{}^{(m)}, \dots, y^{(m)}\right)$
- Model $\hat{y} = w_0 x_0 + w_1 x_1 + \cdots + b$
- Cost $J(\boldsymbol{\theta}) = \dfrac{1}{2m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)^2$
- Gradients

$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) x_0{}^{(i)}$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right) x_1{}^{(i)}$$

$$\dots$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)$$

GD for Multiple Linear Regression

- Vector form - Data (N features, M data)

$$\underset{\text{(M x N)}}{\mathbf{X}} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} & \dots \\ {x_0}^{(2)} & {x_1}^{(2)} & \dots \\ {x_0}^{(3)} & {x_1}^{(3)} & \dots \\ & \dots & \end{bmatrix} \qquad \underset{\text{(M x 1)}}{\mathbf{y}} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \end{bmatrix} \qquad \underset{\text{(N x 1)}}{\mathbf{w}} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \end{bmatrix}$$

- Model

$$\hat{y}^{(1)} = w_0 {x_0}^{(1)} + w_1 {x_1}^{(1)} \dots + \mathrm{b}$$
$$\hat{y}^{(2)} = w_0 {x_0}^{(2)} + w_1 {x_1}^{(2)} \dots + \mathrm{b}$$
$$\hat{y}^{(3)} = w_0 {x_0}^{(3)} + w_1 {x_1}^{(3)} \dots + \mathrm{b}$$
$$\dots$$

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \\ \dots \end{bmatrix} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)}, \dots \\ {x_0}^{(2)} & {x_1}^{(2)}, \dots \\ {x_0}^{(3)} & {x_1}^{(3)}, \dots \\ & \dots \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ w_1 \\ \dots \end{bmatrix} + \mathrm{b}$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

- Cost

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$cost = \frac{1}{2m} sum(\hat{\mathbf{y}} - \mathbf{y})^2$$

$$\left((\mathbf{X}\mathbf{w} + b - \mathbf{y})^T (\mathbf{X}\mathbf{w} + b - \mathbf{y}) \right)$$

GD for Multiple Linear Regression

- Gradient

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}){x_0}^{(1)} + (\hat{y}^{(2)} - y^{(2)}){x_0}^{(2)} + \cdots\right)$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}){x_1}^{(1)} + (\hat{y}^{(2)} - y^{(2)}){x_1}^{(2)} + \cdots\right)$$

$$\ldots$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}) + (\hat{y}^{(2)} - y^{(2)}) + \cdots\right)$$

$$\begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \\ \ldots \end{bmatrix} = \frac{1}{m}\begin{bmatrix} {x_0}^{(1)}, {x_0}^{(2)}, {x_0}^{(3)}, \ldots \\ {x_1}^{(1)}, {x_1}^{(2)}, {x_1}^{(3)}, \ldots \\ \ldots \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \\ \ldots \end{bmatrix}$$

$$\nabla J(\mathbf{w}) = \frac{1}{m}\mathbf{X}^{\mathbf{T}}(\hat{\mathbf{y}} - \mathbf{y})$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \\ \ldots \end{bmatrix}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} sum(\hat{\mathbf{y}} - \mathbf{y})$$

GD for Multiple Linear Regression

- Learning (gradient descent)

Given

$$\mathbf{X} = \begin{bmatrix} x_0{}^{(1)} & x_1{}^{(1)} & \dots \\ x_0{}^{(2)} & x_1{}^{(2)} & \dots \\ x_0{}^{(3)} & x_1{}^{(3)} & \dots \\ & \dots & \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \end{bmatrix}$$

Initialize $\mathbf{w} = [w_1 \quad w_2 \quad \dots],\qquad b$

repeat

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

$$\boxed{\begin{aligned} \mathbf{w} &= \mathbf{w} - \alpha \frac{1}{m}\left(\mathbf{X}^{\mathrm{T}}(\hat{\mathbf{y}} - \mathbf{y})\right) \\ b &= b - \alpha \frac{1}{m}(sum(\hat{\mathbf{y}} - \mathbf{y})) \end{aligned}}$$

$$\mathbf{w} = \mathbf{w} - \alpha\, \nabla J(\mathbf{w})$$

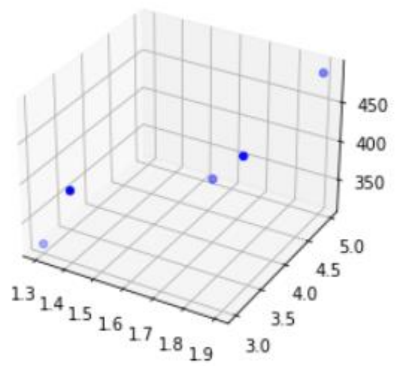
$$b = b - \alpha \frac{\partial J}{\partial b}$$

$$cost = \frac{1}{2m} sum(\hat{\mathbf{y}} - \mathbf{y})^2$$

Example

- Data

```
X_train = np.array([[1.8, 4], [1.3, 3], [1.7, 4], [1.9, 5], [1.4, 3]])
y_train = np.array([430., 320., 390., 490., 400.])
```



- Compute $\hat{y}$ and cost

```
y_hat = np.dot(X, w) + b
cost = np.sum((y_hat - y)**2) / (2*m)
```

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

$$cost = \frac{1}{2m} sum(\hat{\mathbf{y}} - \mathbf{y})^2$$

- Compute gradients

```
y_hat = np.dot(X, w) + b
err = y_hat - y

dj_dw = np.dot(X.T, err) / m
dj_db = np.sum(err) / m
```

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{1}{m} \mathbf{X}^{\mathbf{T}} (\hat{\mathbf{y}} - \mathbf{y})$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} sum(\hat{\mathbf{y}} - \mathbf{y})$$

Example

- Learning (gradient descent)

```
def gradient_descent(X, y, w, b, alpha, num_iters):

    J_history = []
    for i in range(num_iters):
        dj_dw, dj_db = compute_gradient(X, y, w, b)

        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        J_history.append(compute_cost(X, y, w, b))
```

$$\mathbf{w} = \mathbf{w} - \alpha\, \nabla J(\mathbf{w})$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

```
w_init = np.zeros(X_train.shape[1])
b_init = 0.
iterations = 10000
alpha = 0.1

w_final, b_final, J_hist = gradient_descent(X_train ,y_train, w_init, b_init, alpha, iterations)
```

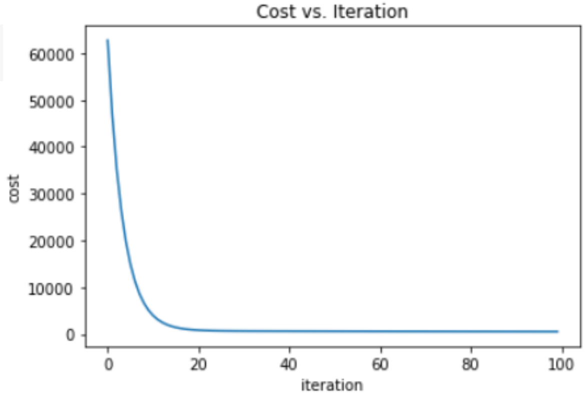
```
Iteration     0: Cost 6.27e+04
Iteration  1000: Cost 3.76e+02
Iteration  2000: Cost 3.74e+02
Iteration  3000: Cost 3.73e+02
Iteration  4000: Cost 3.73e+02
Iteration  5000: Cost 3.72e+02
```

```
w = [137.51483137  22.57217992]
b = 97.45385409256511
```

Example

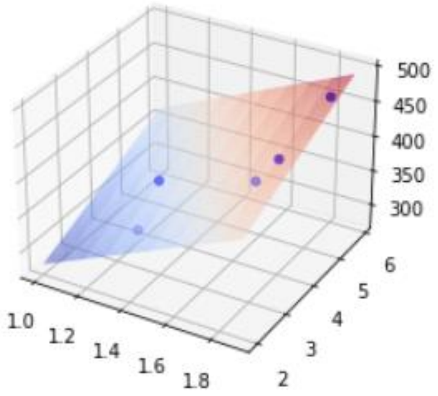
- Cost change

```
plt.plot(J_hist[:100])
```



- Learned model

```
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

ax.scatter(X_train[:,0], X_train[:,1], y_train, marker='o', c='blue')

X = np.arange(1, 2, 0.1)
Y = np.arange(2, 6, 0.1)
X, Y = np.meshgrid(X, Y)

Z = w_final[0]*X + w_final[1]*Y + b_final
ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, alpha=0.5)
```



Different notation

- Different notation (ex> Python ML book)

$$\mathbf{x} = [x_0 \quad x_1]$$

$$\mathbf{w} = [w_0 \quad w_1], \qquad b$$

$$\hat{y} = w_0 x_0 + w_1 x_1 + b$$

$$\mathbf{X} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} \\ {x_0}^{(2)} & {x_1}^{(2)} \\ {x_0}^{(3)} & {x_1}^{(3)} \\ & \cdots \end{bmatrix}$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

$$\mathbf{x} = [1 \quad x_1 \quad x_2]$$

$$\mathbf{w} = [w_0 \quad w_1 \quad w_2] \quad (w_0 = b)$$

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$\mathbf{X} = \begin{bmatrix} 1 & {x_1}^{(1)} & {x_2}^{(1)} \\ 1 & {x_1}^{(2)} & {x_2}^{(2)} \\ 1 & {x_1}^{(3)} & {x_2}^{(3)} \\ & \cdots & \end{bmatrix}$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

Closed Form Solution

- Cost function

$$J(\boldsymbol{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{Xw} - \mathbf{y}\|^2 = (\mathbf{Xw} - \mathbf{y})^T(\mathbf{Xw} - \mathbf{y})$$

$$\|\mathbf{y}\| = \left(\sum \left(y^{(i)}\right)^2\right)^{\frac{1}{2}}$$

- Minimize ***w***

$$\frac{\partial J}{\partial \mathbf{w}} = 2\,\mathbf{X}^{\mathbf{T}}(\mathbf{Xw} - \mathbf{y}) = \mathbf{0}$$

$$\mathbf{X}^{\mathbf{T}}\mathbf{Xw} = \mathbf{X}^{\mathbf{T}}\mathbf{y}$$

$$\therefore \mathbf{w} = \left(\mathbf{X}^{\mathbf{T}}\mathbf{X}\right)^{-\mathbf{1}}\mathbf{X}^{\mathbf{T}}\mathbf{y}$$

Normal equation

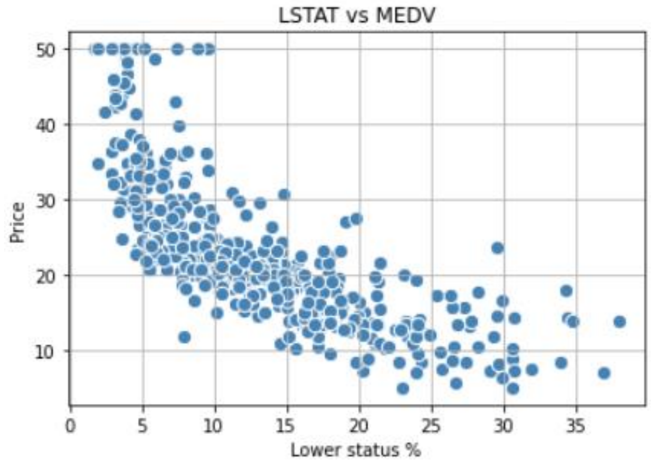
- Solving w using normal equation
  - Only for linear regression - can not be applied to other learning algorithms
  - Solving the normal equation can be very difficult for large N and M

Linear Regression using Scikit Learn

- The Boston housing dataset

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

- RM — average number of rooms per dwelling
- LSTAT — percentage of lower status of the population
- MEDV — median value of owner-occupied homes



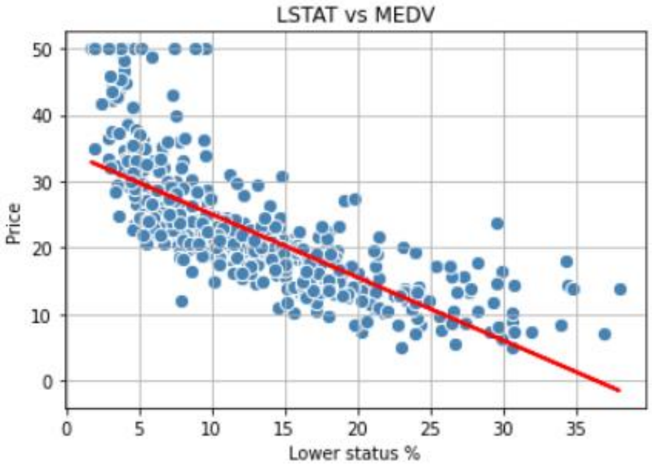
Linear Regression using Scikit Learn

- Use sklearn.linear_model.LinearRegression. Training the model - .fit()

```
from sklearn.linear_model import LinearRegression

# training the model
lr = LinearRegression()
lr.fit(X, y)
```

```
# model parameters
print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

```
w =  [[ 9.10210898]]
b =  [-34.67062078]
```



- Predicting values - .predict( )

```
y_hat = lr.predict(X)

# print the MSE
print('MSE : %.2f' % mean_squared_error(y, y_hat))
```

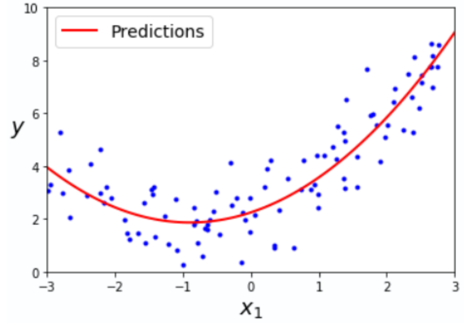
```
MSE : 38.48
```

Polynomial Regression

- The model
  - y = f(x) is modeled as a polynomial function

$$y = f_{\theta}(x) = w_0 \cdot x + w_1 \cdot x^2 + w_2 \cdot x^3 + \ldots + b$$

parameters
(w_0 , w_1 , b)



  - We can use multiple linear regression
    - Introduce new features
    - Data

$$(x, y) \quad \Rightarrow \quad (x \ , x^2, x^3, \ldots, y)$$

    - Model

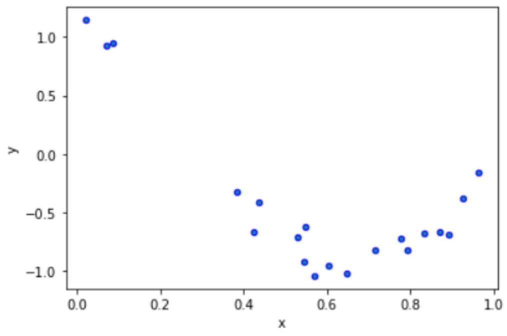
$$\hat{y} = w_0 x + w_1 x^2 + \cdots + b$$

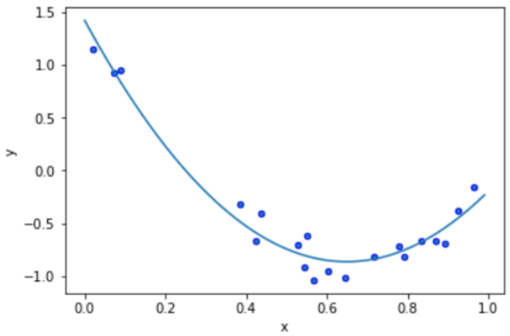
Regularization

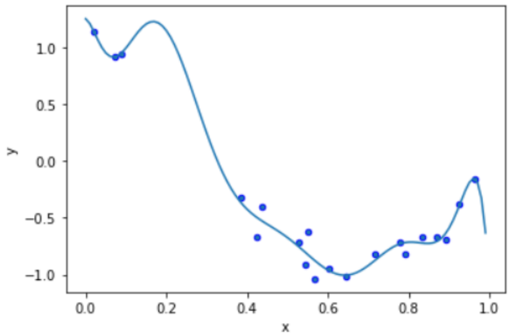
- Overfitting
  - When the model is too complex
    → good for training data, but bad for test data
- Example

$y = w_2x^2 + w_1x^1 + w_0$

$y = w_{10}x^{10} + w_9x^9 + \cdots + w_1x^1 + w_0$





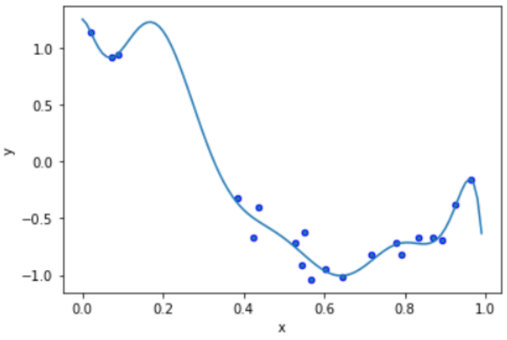


Regularization

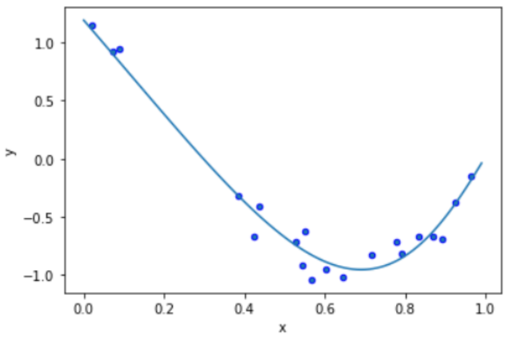
- Regularization
  - Reduce the size of parameters w

    → make the model simpler
- Example

$$y = w_{10}x^{10} + w_9x^9 + \cdots + w_1x^1 + w_0$$



$$y = w_{10}x^{10} + w_9x^9 + \cdots + w_1x^1 + w_0$$



Regularization

- How to reduce w?
  - Add regularization term(*penalty*) to a cost function

    Ex>

    $$Cost\ Function = J(w) + \frac{1}{2m}\lambda\sum_{j=1}^{n} w_j^2$$

    - λ : regularization parameter
  - Then, minimizing cost = reduce w → reduce complexity
  - Gradient descent

    $$w_j = w_j - \alpha\frac{\partial J}{\partial w_j} - \alpha\frac{\lambda}{m}w_j$$

  - If $\lambda = 0$, the model overfits $(y = w_{10}x^{10} + w_9x^9 + \cdots + w_1x^1 + w_0)$

    If λ is too large, the model underfits $(y = w_1x^1 + w_0)$

Polynomial Regression using Scikit Learn

- Extending x → x, x^2, x^3, …
  - sklearn.preprocessing.PolynomialFeatures

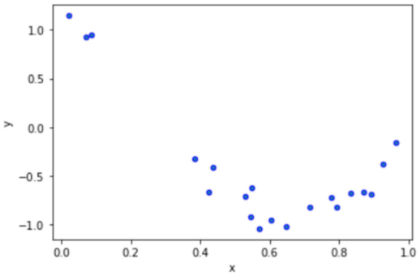
```
from sklearn.preprocessing import PolynomialFeatures

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

```
[[0.0202184 ]   ➡   [[0.0202184  0.00040878]
```

- Regularizing linear regression
  - sklearn.linear_model.Ridge
    - Cost = $||\mathbf{Xw} - \mathbf{y}||^2 + \lambda||\mathbf{w}||^2$
    - 'alpha' is the regularization parameter

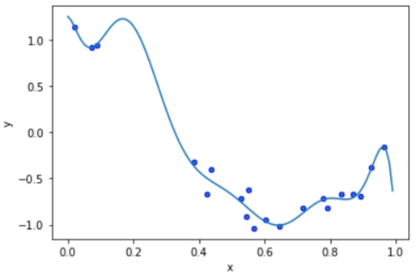
Polynomial Regression using Scikit Learn

- Data
  - (x,   cos(x)+random noise)



- Polynomial regression with degree=10

```
lr = LinearRegression()
lr.fit(X_poly, y)

X_test = np.arange(0, 1, 0.01).reshape(-1, 1)
X_test_poly = poly.fit_transform(X_test)

plt.plot(X_test, lr.predict(X_test_poly))
```



- Regularization – use Ridge with λ

```
lr = Ridge(alpha = 0.01)
lr.fit(X_poly, y)

X_test = np.arange(0, 1, 0.01).reshape(-1, 1)
X_test_poly = poly.fit_transform(X_test)

plt.plot(X_test, lr.predict(X_test_poly))
```