Introduction to Regression 2

Hee-il Hahn

Professor
Department of Information and Communications Engineering
Hankuk University of Foreign Studies
hihahn@hufs.ac.kr

2-parameter linear regression

Observable dataset : $\mathbf{d}_1(x_1, y_1), \mathbf{d}_2(x_2, y_2) \dots \mathbf{d}_n(x_n, y_n)$

Model : $y = wx + b$

Compute mean squared error of the model on the dataset

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - wx_i - b)^2$$

To minimize MSE

$$\begin{cases} \frac{\partial}{\partial w} MSE = \frac{\partial}{\partial w}\frac{1}{n}\sum_{i=1}^{n}(y_i - wx_i - b)^2 = 0 \ \rightarrow \sum_{i=1}^{n} x_i\,(y_i - wx_i - b) = 0 \\ \frac{\partial}{\partial b} MSE = \frac{\partial}{\partial b}\frac{1}{n}\sum_{i=1}^{n}(y_i - wx_i - b)^2 = 0 \ \rightarrow \sum_{i=1}^{n}(y_i - wx_i - b) = 0 \end{cases}$$

$$\Rightarrow \begin{cases} w\sum_{i=1}^{n} {x_i}^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ w\sum_{i=1}^{n} x_i + nb = \sum_{i=1}^{n} y_i \end{cases} \qquad w = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} {x_i}^2 - (\sum_{i=1}^{n} x_i)^2}, \quad b = \frac{\sum_{i=1}^{n} y_i - w\sum_{i=1}^{n} x_i}{n},$$

2-parameter linear regression – cont.

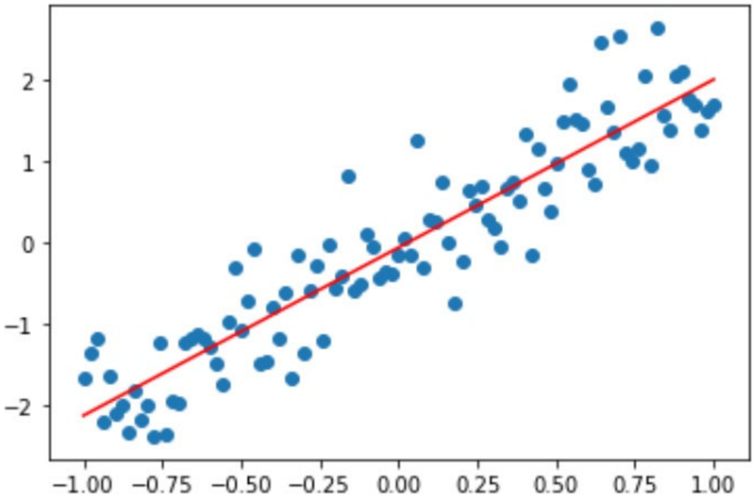
- Simulation(Python)

```python
import numpy as np
import matplotlib.pyplot as plt        # 그래프를 그리기 위해 불러온다.

num_data = 101                                          # 101개의 데이터를 생성한다
x_train = np.linspace(-1, 1, num_data)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.5

w = (len(x_train) * np.sum(x_train*y_train) - np.sum(x_train) * np.sum(y_train))  /

b = (np.sum(y_train) - w * np.sum(x_train) ) / len(x_train)
```

executed in 16ms, finished 19:41:54 2020-03-25

```python
def model(X, w, b):
    return tf.add(tf.multiply(X, w), b)

plt.scatter(x_train, y_train)
y_learned = w * x_train + b
plt.plot(x_train, y_learned, 'r' )
plt.show()
```

executed in 367ms, finished 19:42:02 2020-03-25



2-parameter linear regression – cont.

- Simulation(TensorFlow) Model : $y = wx$

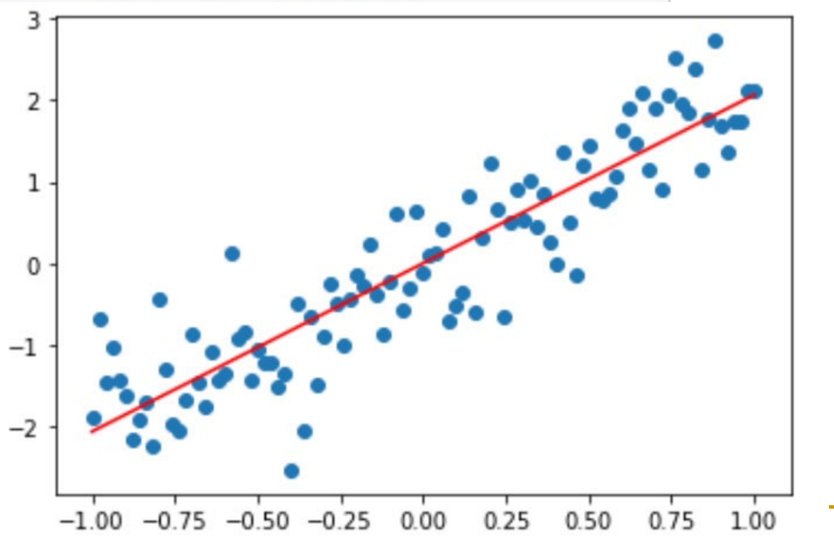
```
import tensorflow as tf  # 학습 알고리즘을 위해 tensorflow를 불러온다.
import numpy as np       # 데이터 초기화를 위해 numpy를 불러온다.
import matplotlib.pyplot as plt       # 그래프를 그리기 위해 불러온다.

learning_rate = 0.01         # 학습 알고리즘이 사용할 상수(hyper parameters) 정의
training_epochs = 100
x_train = np.linspace(-1, 1, 101)
y_train = 3 * x_train + 5 + np.random.randn(*x_train.shape) * 0.5
X = tf.placeholder(tf.float32)     # 입력노드와 출력노드를 placeholder로 설정함으로써
Y = tf.placeholder(tf.float32)     # 실제로 값은 x_train과 y_train에 의해 입력받도록

def model(X, w, b):
    return tf.add(tf.multiply(X, w), b)

w = tf.Variable(tf.random_uniform([1], -1, 1), name = "weight0")    # 가중치 변수를
b = tf.Variable(0.0, name = "weight1")

y_model = model(X, w, b)                # cost function을 정의한다.
cost = tf.reduce_mean(tf.square(Y-y_model))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

2-parameter linear regression – cont.

- Simulation(TensorFlow) Model : $y = wx$

```
sess = tf.Session()                    # 세션을 설정하고 모든 변수를 초기화한다.
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    for (x,y) in zip(x_train, y_train):
        sess.run(train_op, feed_dict={X: x, Y: y})        # cost function을 minimize

w_val = sess.run(w)                    # 최종 파라미터 값을 얻어낸다.

sess.close()

plt.scatter(x_train, y_train)
y_learned = x_train*w_val
plt.plot(x_train, y_learned, 'r' )
plt.show()
```

executed in 2.96s, finished 17:04:39 2020-03-25



2-parameter linear regression – cont.

- Simulation(TensorFlow) Model : $y = wx + b$

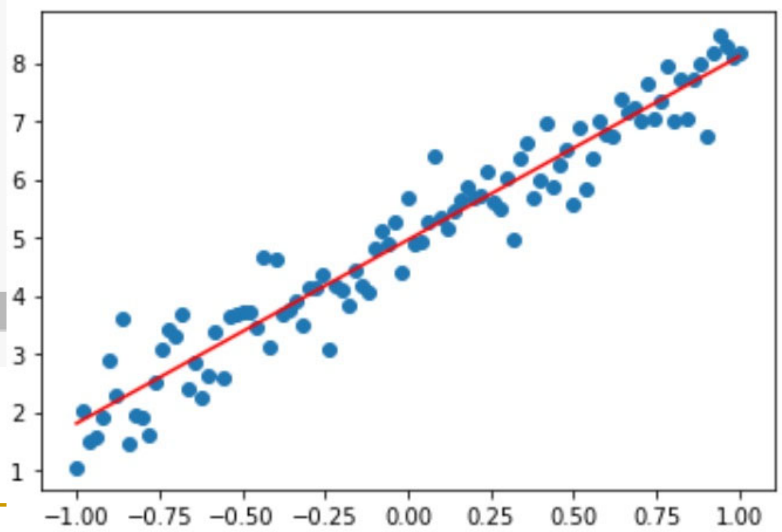
```
import tensorflow as tf  # 학습 알고리즘을 위해 tensorflow를 불러온다.
import numpy as np       # 데이터 초기화를 위해 numpy를 불러온다.
import matplotlib.pyplot as plt      # 그래프를 그리기 위해 불러온다.

learning_rate = 0.01        # 학습 알고리즘이 사용할 상수(hyper parameters) 정의
training_epochs = 100
x_train = np.linspace(-1, 1, 101)
y_train = 3 * x_train + 5 + np.random.randn(*x_train.shape) * 0.5
X = tf.placeholder(tf.float32)     # 입력노드와 출력노드를 placeholder로 설정함으로써
Y = tf.placeholder(tf.float32)     # 실제로 값은 x_train과 y_train에 의해 입력받도록

def model(X, w, b):
    return tf.add(tf.multiply(X, w), b)

w = tf.Variable(tf.random_uniform([1], -1, 1), name = "weight0")     # 가중치 변수를
b = tf.Variable(0.0, name = "weight1")

y_model = model(X, w, b)                # cost function을 정의한다.
cost = tf.reduce_mean(tf.square(Y-y_model))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

2-parameter linear regression – cont.

- Simulation(TensorFlow) Model : $y = wx + b$

```
sess = tf.Session()                  # 세션을 설정하고 모든 변수를 초기화한다.
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    for (x,y) in zip(x_train, y_train):
        sess.run(train_op, feed_dict={X: x, Y: y})        # cost function을 minimiz

w_val = sess.run(w)                  # 최종 파라미터 값을 얻어낸다.
b_val = sess.run(b)

sess.close()

plt.scatter(x_train, y_train)
y_learned = x_train * w_val + b_val
plt.plot(x_train, y_learned, 'r' )
plt.show()
```

executed in 3.74s, finished 19:06:03 2020-03-25



Multivariate regression

- What if the inputs are vectors?

  Write matrix $\boldsymbol{X}$ and $\mathbf{y}$:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$$

  where $\mathbf{x}_1 = (x_{11}, \cdots, x_{1p}), \mathbf{x}_2 = (x_{21}, \cdots, x_{1p}), \cdots$

$$\mathbf{x}_n = (x_{n1}, \cdots, x_{np})$$

- Assume that the data is formed by $y_i = \boldsymbol{w}^T \mathbf{x}_i + noise_i$

$$y \sim N(\boldsymbol{w}^T \mathbf{x}, \sigma^2)$$

Multivariate regression - cont.

- Probability of each response variable

$$P(\mathbf{d}_i|\mathbf{w}) = P(y_i|\mathbf{w},\mathbf{x}_i) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2}\left(\frac{y_i-\mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2\right).$$

- Given data $\boldsymbol{D} = \{\mathbf{d}_1(\mathbf{x}_1, y_1), \cdots, \mathbf{d}_n(\mathbf{x}_n, y_n)\}$, we want to estimate the weight vector $\mathbf{w}$.

Likelihood:

$$L(\mathbf{w}) = P(\boldsymbol{D}|\mathbf{w}) = P(y|\mathbf{w},\mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{d}_i|\mathbf{w})$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2}\left(\frac{y_i-\mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2\right)$$

Log-likelihood:

$$logL(\mathbf{w}) = \sum_{i=1}^{n}\left\{-\frac{1}{2}log(2\pi\sigma^2) - \frac{1}{2}\left(\frac{y_i-\mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2\right\}$$

Multivariate regression - cont.

- Maximum likelihood solution:

$$\hat{\mathbf{w}}_{MLE} = \text{argmax}_{\mathbf{w}} \prod_{i=1}^{n} P(\mathbf{d}_i|\mathbf{w})$$

$$= \text{argmax}_{\mathbf{w}} \sum_{i=1}^{n} -\frac{1}{2}\left(\frac{y_i - \mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2$$

$$= \text{argmin}_{\mathbf{w}} \sum_{i=1}^{n} \frac{1}{2}\left(\frac{y_i - \mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2 \quad \Leftarrow \left\{\sum_{i=1}^{n} \frac{1}{2}\left(\frac{y_i - \mathbf{w}^T\mathbf{x}_i}{\sigma}\right)^2 = (\boldsymbol{y} - \boldsymbol{X}\mathbf{w})^T(\boldsymbol{y} - \boldsymbol{X}\mathbf{w})\right\}$$

$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}$$

(from $\frac{d}{d\mathbf{w}}(\boldsymbol{y} - \boldsymbol{X}\mathbf{w})^T(\boldsymbol{y} - \boldsymbol{X}\mathbf{w}) = \mathbf{0}$)

Multivariate regression - cont.

- $\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}$ $\quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$

- $\boldsymbol{X}\mathbf{w} = \begin{bmatrix} x_{11}w_1 + x_{12}w_2 \cdots + x_{1n}w_n \\ \vdots \\ x_{n1}w_1 + x_{n2}w_2 \cdots + x_{nn}w_n \end{bmatrix}$ $\quad \boldsymbol{y} - \boldsymbol{X}\mathbf{w} = \begin{bmatrix} y_1 - x_{11}w_1 - x_{12}w_2 \cdots - x_{1n}w_n \\ \vdots \\ y_n - x_{n1}w_1 - x_{n2}w_2 \cdots - x_{nn}w_n \end{bmatrix}$

- $\boldsymbol{f}(\mathbf{w}) = (\boldsymbol{y} - \boldsymbol{X}\mathbf{w})^T(\boldsymbol{y} - \boldsymbol{X}\mathbf{w})$
- $= (y_1 - x_{11}w_1 - x_{12}w_2 \cdots - x_{1n}w_n)^{\mathbf{2}} + \cdots + (y_n - x_{n1}w_1 - x_{n2}w_2 \cdots - x_{nn}w_n)^{\mathbf{2}}$
- $= \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$

Multivariate regression - cont.

- $\frac{d}{d\mathbf{w}}\boldsymbol{f} = \begin{bmatrix} \frac{f}{dw_1} \\ \vdots \\ \frac{f}{dw_1} \end{bmatrix} = \begin{bmatrix} -2\sum_{i=1}^{n} x_{i1}(y_i - \mathbf{w}^T\mathbf{x}_i) \\ \vdots \\ -2\sum_{i=1}^{n} x_{in}(y_i - \mathbf{w}^T\mathbf{x}_i) \end{bmatrix} = 0 \quad \boldsymbol{X}^T\mathbf{y} = \begin{bmatrix} \sum_{i=1}^{n} x_{i1}y_i \\ \vdots \\ \sum_{i=1}^{n} x_{in}y_i \end{bmatrix} \boldsymbol{X}^T\boldsymbol{X} = \begin{bmatrix} \sum_{i=1}^{n} x_{i1}\mathbf{x}_i \\ \vdots \\ \sum_{i=1}^{n} x_{in}\mathbf{x}_i \end{bmatrix}$

$$\mathbf{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}$$

(from $\frac{d}{d\mathbf{w}}(\boldsymbol{y} - \boldsymbol{X}\mathbf{w})^T(\boldsymbol{y} - \boldsymbol{X}\mathbf{w}) = \mathbf{0}$)

5^{th}-degree polynomial regression

- Simulation(TensorFlow) Model : $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_5 x^5$

```
import tensorflow as tf  # 학습 알고리즘을 위해 tensorflow를 불러온다.
import numpy as np       # 데이터 초기화를 위해 numpy를 불러온다.
import matplotlib.pyplot as plt      # 그래프를 그리기 위해 불러온다.

learning_rate = 0.01       # 학습 알고리즘이 사용할 상수(hyper parameters) 정의
training_epochs = 40

x_train = np.linspace(-1, 1, 101)
num_coeffs = 6
y_train_coeffs = [1, 2, 3, 4, 5, 6]

y_train = 0
for i in range(num_coeffs):
    y_train += y_train_coeffs[i] * np.power(x_train, i)

y_train +=  np.random.randn(*x_train.shape) * 2
X = tf.placeholder(tf.float32)     # 입력노드와 출력노드를 placeholder로 설정함으로써
Y = tf.placeholder(tf.float32)     # 실제로 값은 x_train과 y_train에 의해 입력받도록
```

5^{th}-degree polynomial regression – cont.

- Simulation(TensorFlow) Model : $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_5 x^5$

```
def model(X, w):
    terms = []
    for i in range(num_coeffs):
        term = tf.multiply(w[i], tf.pow(X, i))
        terms.append(term)
    return tf.add_n(terms)

w = tf.Variable([0.0] * num_coeffs, name = "parameters")    # 가중치 변수를 설정한다

y_model = model(X, w)              # cost function을 정의한다.
cost = tf.square(Y-y_model)
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

executed in 323ms, finished 20:11:59 2020-03-25

5^{th}-degree polynomial regression – cont.

- Simulation(TensorFlow)   Model : $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_5 x^5$

```
sess = tf.Session()                    # 세션을 설정하고 모든 변수를 초기화한다.
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    for (x,y) in zip(x_train, y_train):
        sess.run(train_op, feed_dict={X: x, Y: y})

w_val = sess.run(w)                      # 최종 파라미터

print(w_val)

sess.close()

plt.scatter(x_train, y_train)

y_learned = 0
for i in range(num_coeffs):
    y_learned += w_val[i] * np.power(x_train, i)
plt.plot(x_train, y_learned, 'r' )
plt.show()
```

```
[0.7066829 2.0447004 4.475396  4.3324866 3.5885096 4.7185984]
```