02.27 스터디

주제: 선형 회귀

▼ 챕터

1. 문제의 정의
2. 이 장의 중요 개념
3. 선형 함수(nn.Linear)
4. 커스텀 클래스를 이용한 모델 정의
5. MSELoss 클래스를 이용한 손실 함수
6. 데이터 준비
7. 모델 정의
8. 경사 하강법
9. 결과 확인
10. 중회귀 모델로 확장
11. 학습률의 변경

선형회귀

독립 변수와 종속 변수 간의 관계를 선형 방정식으로 모델링하며, 가중치, 편향를 학습

- 지도학습
- 회귀 → 목적변수가 연속형

단순회귀: Feature가 하나

중회귀: Feature가 여러개

다항회귀: 2차항 이상이 포함된 경우

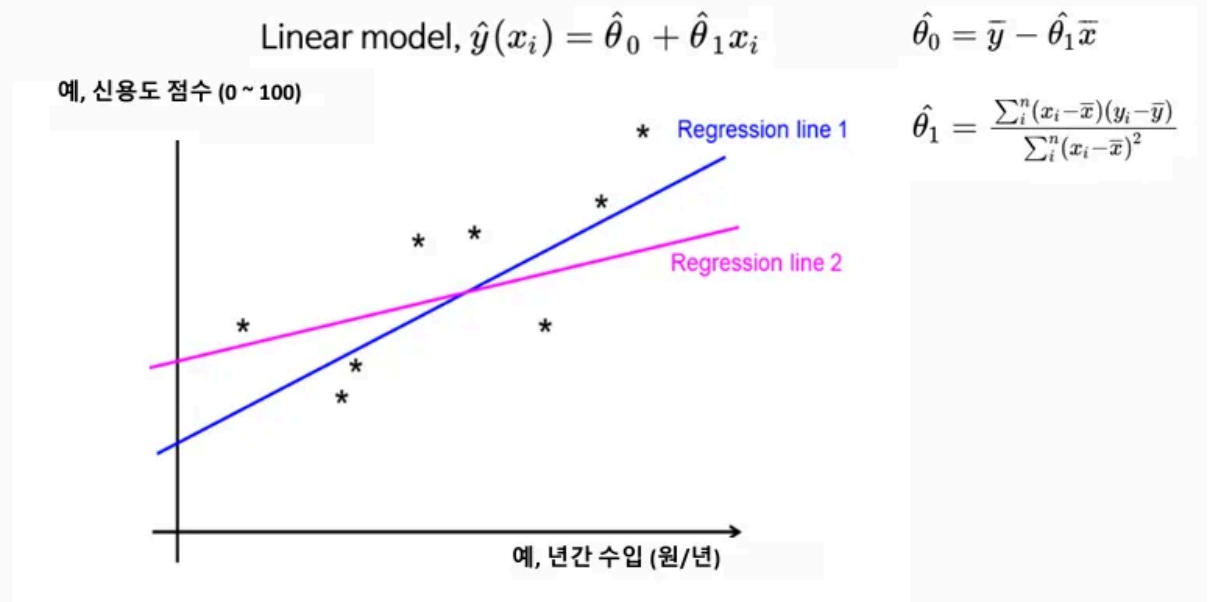
주요 선형회귀 기법

1. OLS (Ordinary Least Squares, 최소제곱법)
2. Ridge Regression (릿지 회귀)

3. Lasso Regression (라쏘 회귀)
4. Elastic Net (엘라스틱넷 회귀)
5. Robust Regression (강건 회귀)
6. Bayesian Regression (베이지안 회귀)

최소 제곱 추정 (OLS)

잔차(residual, 오차)의 제곱합을 최소화하는 방식으로 최적의 회귀선을 구하는 기법



데이터를 가장 잘 설명하는 / 정답을 잘 예측하는 회귀선은??

회귀선과 정답의 차이 e_1, e_2, ..., e_n을 모두 합한 값이 최소화 되는 직선을 찾는다.
⇒ 이런 접근법의 문제점은?

에러를 단순히 더하기만 하면 (양수, 음수가 같이 더해져서) 에러 상쇄되는 문제 발생

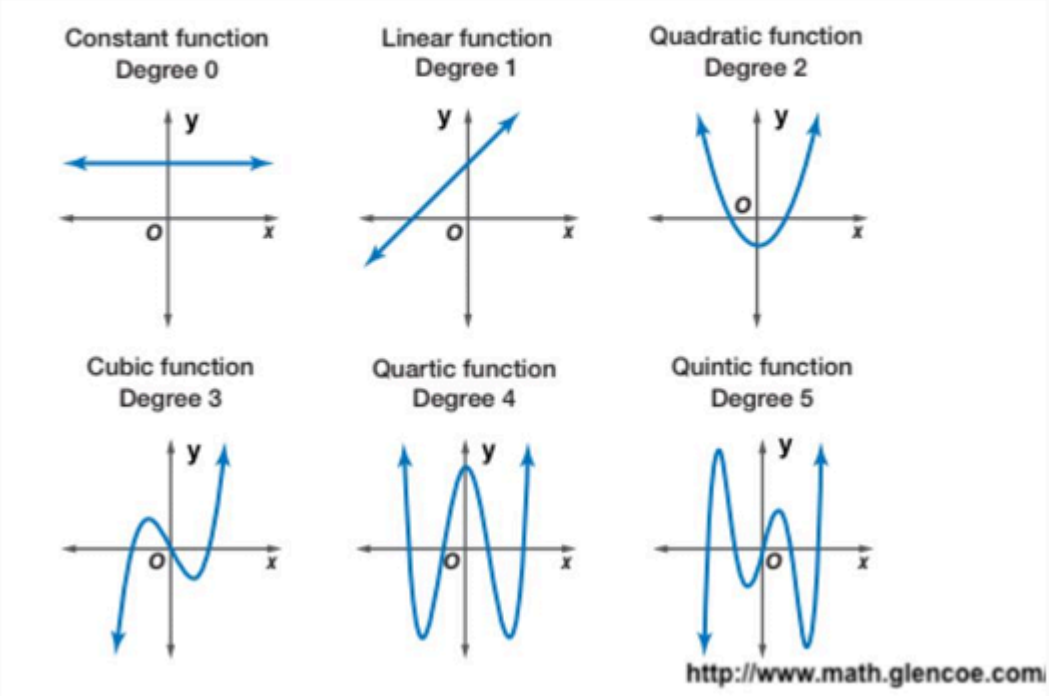
해경방법 - ①절대값 / ②제곱하기
②제곱하는 방법이 다루기 편하고 미분가능해진다는 장점이 있다.

에러값을 모두 제곱해서 더한 다음 에러값의 합이 최소가 되는 직선을 찾는다.

다항회귀

모델이 복잡해질 수록 차수(degree) 증가

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_p x^p + \epsilon_i$$



주요 메서드 & 속성

메서드/속성	설명
l1.parameters() l1.named_parameters()	모델의 파라미터 값(가중치, 편향)만 반환 파라미터의 이름과 값(가중치, 편향)을 함께 반환
l1.weight l1.bias	입력 데이터에 곱해지는 학습 가능한 가중치 행렬 각 출력 뉴런에 더해지는 편향 값
tensor.shape tensor.size()	텐서의 크기(차원) 반환 `torch.Size` 객체
nn.init.constant_()	텐서를 지정한 상수 값으로 초기화
nn.MSELoss()	PyTorch의 손실 함수 MSE 생성자
nn.Linear()	선형 변환 레이어 생성

선형함수 정의

```
# 입력 텐서
x_np = np.array([[0, 0], [0, 1], [1, 0], [1,1]])
x =  torch.tensor(x_np).float()

# 선형 레이어 생성
l3 = nn.Linear(2, 3)
print("----------------------초기화 전:")
print(l3.weight)    #3×2텐서
print(l3.bias)



# 초깃값 설정
nn.init.constant_(l3.weight[0,:], 1.0)  # 1행의 모든 열을 1.0으로
nn.init.constant_(l3.weight[1,:], 2.0)
nn.init.constant_(l3.weight[2,:], 3.0)
nn.init.constant_(l3.bias, 2.0)

# 결과 확인
print("----------------------초기화 후:")
print(l3.weight)
print(l3.bias)

# 함수 값 계산
print("----------------------입력데이터 통과:")
y3 = l3(x2)
print(y3.shape)
print(y3.data)

<출력>
----------------------초기화 전:
Parameter containing:
tensor([[ 0.0636,  0.0693],
        [-0.2145,  0.0034],
        [-0.2199,  0.2032]], requires_grad=True)
```

```
Parameter containing:
tensor([ 0.6867,  0.1071, -0.3132], requires_grad=True)

----------------------초기화 후:
Parameter containing:
tensor([[1., 1.],
        [2., 2.],
        [3., 3.]], requires_grad=True)
Parameter containing:
tensor([2., 2., 2.], requires_grad=True)

----------------------입력데이터 통과:
torch.Size([4, 3])
tensor([[2., 2., 2.],
        [3., 4., 5.],
        [3., 4., 5.],
        [4., 6., 8.]])
```

클래스를 사용한 모델 정의 - 입력1, 출력1(히든레이어 x)

```
## 모델 정의
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()

        self.l1 = nn.Linear(n_input, n_output)

    # 예측 함수
    def forward(self, x):
        x1 = self.l1(x) # 선형 회귀
        return x1

# 더미 입력
inputs = torch.rand(100,1)
labels1 = torch.rand(100,1)
```

```
# 인스턴스 생성( 1 입력, 1 출력 선형 모델)
n_input = 1
n_output = 1
net = Net(n_input, n_output)

## 예측
outputs = net(inputs)
print("outputs = \n", outputs)

## 손실함수
criterion = nn.MSELoss()
loss = criterion(outputs, labels1)
print("loss = ", loss)
loss.backward()

print(net.l1.weight.grad)
print(net.l1.bias.grad)
```

보스턴 주택가격 데이터셋(Boston Housing Dataset)

데이터 불러오기

```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+",skiprows=22, header=None)
print(raw_df.head(10))
```

```
<출력>
0      1      2      3      4    5      6      7      8       9     10
0    0.00632  18.00   2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  396.90000   4.98  24.00  NaN    NaN    NaN   NaN     NaN  NaN    NaN   NaN
2    0.02731   0.00   7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
3  396.90000   9.14  21.60  NaN    NaN    NaN   NaN     NaN  NaN    NaN   NaN
4    0.02729   0.00   7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
5  392.83000   4.03  34.70  NaN    NaN    NaN   NaN     NaN  NaN    NaN   NaN
```

```
6    0.03237   0.00   2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
7  394.63000   2.94  33.40  NaN   NaN    NaN   NaN    NaN  NaN   NaN   NaN
8    0.06905   0.00   2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7
9  396.90000   5.33  36.20  NaN   NaN    NaN   NaN    NaN  NaN   NaN   NaN
```

행 합치기

```
x_org = np.hstack([raw_df.values[::2, :],
                   raw_df.values[1::2, :2]]) # 짝수줄 전체, 홀 수 줄 [: 2] => Features
```

보스턴 주택 가격 데이터는 일반적인 테이블 형식이 아니라 2줄에 걸쳐 하나의 데이터가 표현됨
홀수번째 행에 있는 2개 데이터를 짝수행과 합치기

shape(1012, 11) →(506, 13)

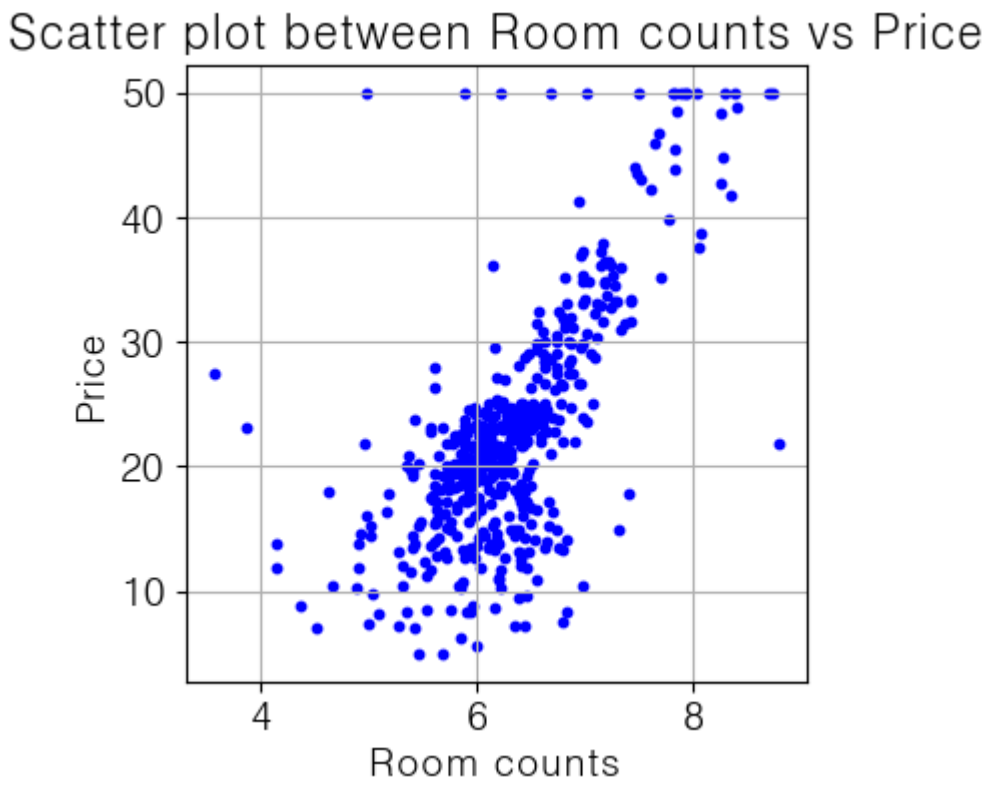
특정feature와 target의 산점도 확인

```
yt = raw_df.values[1::2, 2] ## Target: MEDV

feature_names = np.array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX',
                          'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B', 'LSTAT'])

x = x_org[:,feature_names == 'RM']

# 산포도 출력
plt.scatter(x, yt, s=10, c='b')
plt.xlabel('Room counts')
plt.ylabel('Price')
plt.title(' Scatter plot between Room counts vs Price ')
plt.show()
```



(번외)데이터 프레임에 칼럼명 붙이기

```
raw_df = pd.read_csv(data_url, sep="\s+",
                     skiprows=22, header=None)



df_features = raw_df.iloc[::2, :].reset_index(drop=True)
df_extra = raw_df.iloc[1::2, :3].reset_index(drop=True)
df = pd.concat([df_features, df_extra], axis=1)

# 컬럼명 설정
column_names = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX",
    "PTRATIO", "B", "LSTAT", "MEDV"
]
df.columns = column_names

# RM(방 개수)와 MEDV(주택 가격) 데이터 추출
x = df["RM"].values
y = df["MEDV"].values
```

```
# 산점도 그리기
plt.scatter(x, y, s=10, c='b')
plt.xlabel('Room counts')
plt.ylabel('Price')
plt.title('Scatter plot between Room counts vs Price')
plt.show()
```

```
df = pd.DataFrame([[1, 2], [3, 4]], columns=['X', 'Y'])
```

경사 하강법을 이용한 학습

```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()

        self.l1 = nn.Linear(n_input, n_output)

    def forward(self, x):
        x1 = self.l1(x)
        return x1

#입력값
inputs = torch.tensor(x, dtype = torch.float32)
x = x_org[:,feature_names == 'RM']

net = Net(n_input, n_output) # 모델 정의
criterion = nn.MSELoss() # 손실 함수

lr = 0.01
num_epochs = 50000
optimizer = optim.SGD(net.parameters(), lr=lr)

history = np.zeros((0,2))
```

```
for epoch in range(num_epochs):

    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, labels1)
    loss.backward()
    optimizer.step()

    if epoch % 100 == 0:
        history = np.vstack((history, np.array([epoch, loss.item()])))
        print(f'Epoch {epoch} loss: {loss.item():.5f}')
```

중회귀 모델(다중 선형 회귀)

두 개 이상의 독립 변수를 사용하여 타겟 예측

```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()

        self.l1 = nn.Linear(n_input, n_output)

    def forward(self, x):
        x1 = self.l1(x)
        return x1

# RM & LSTAT
column_names = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX",
    "PTRATIO", "B", "LSTAT", "MEDV"
]
vars = (feature_names=='RM') | (feature_names == 'LSTAT')
x = x_org[:,vars]



n_input = x2.shape[1]
net = Net(n_input, n_output)
```

```
inputs = torch.tensor(x2, dtype = torch.float32)

lr = 0.001
net = Net(n_input, n_output)
criterion = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=lr)
num_epochs = 50000

history2 = np.zeros((0,2))

for epoch in range(num_epochs):

    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, labels1)

    loss.backward()
    optimizer.step()

    if ( epoch % 100 == 0):
        history2 = np.vstack((history2, np.array([epoch, loss.item()])))
        print(f'Epoch {epoch} loss: {loss.item():.5f}')
```

전체 변수를 사용

다중 퍼셉트론으로 모델 수정

```
# 모델 수정
class Net(nn.Module):
    def __init__(self, n_input):
        super().__init__()

        self.l1 = nn.Linear(n_input, 26)
        self.l2 = nn.Linear(26, 48)
        self.l3 = nn.Linear(48, 1)
        self.relu = nn.ReLU()
```

```
    def forward(self, x):
        x1 = self.relu(self.l1(x))
        x2 = self.relu(self.l2(x1))
        x3 = self.l3(x2)

        return x3


x3 = x_org
input = torch.tensor(x3).float()

n_input = x3.shape[1]
net3 = Net(n_input) #처음에 빈칸으로 놔야하나

inputs = torch.tensor(x3, dtype = torch.float32)

lr = 0.001
criterion = nn.MSELoss()
optimizer = optim.SGD(net3.parameters(), lr=lr)
num_epochs = 50000

history3 = np.zeros((0,2))

for epoch in range(num_epochs):

    optimizer.zero_grad()
    outputs = net3(inputs)
    loss = criterion(outputs, labels1)

    loss.backward()
    optimizer.step()

    if epoch % 100 == 0:
        history3 = np.vstack((history3, np.array([epoch, loss.item()])))
        print(f'Epoch {epoch} loss: {loss.item():.5f}')
```