22기 정규세션

ToBig's 21기 강의자
박민지
ToBig's 22기 강의자
최해원

Loss Function

Contents

Unit 01 | Training process using MLE

- Loss function의 개념
- MSE의 개념
- MSE vs MAE vs RMSE
- CV의 개념과 실습

Unit 01 | MSE - Loss Function의 개념

Definition

모델이 예측한 값과 실제값의 차이를 나타내는데 사용되는 함수로,
Target에 대한 예측 정확도를 높이기 위해 함수의 parameter 학습에 사용됨

여기서의 학습은?

target y와 input x에 대해

$$y \approx f(x)$$

즉, target에 근사하는 함수 f(x)를 찾는 것.

Unit 01 | MSE - MSE의 개념 (1)

Definition

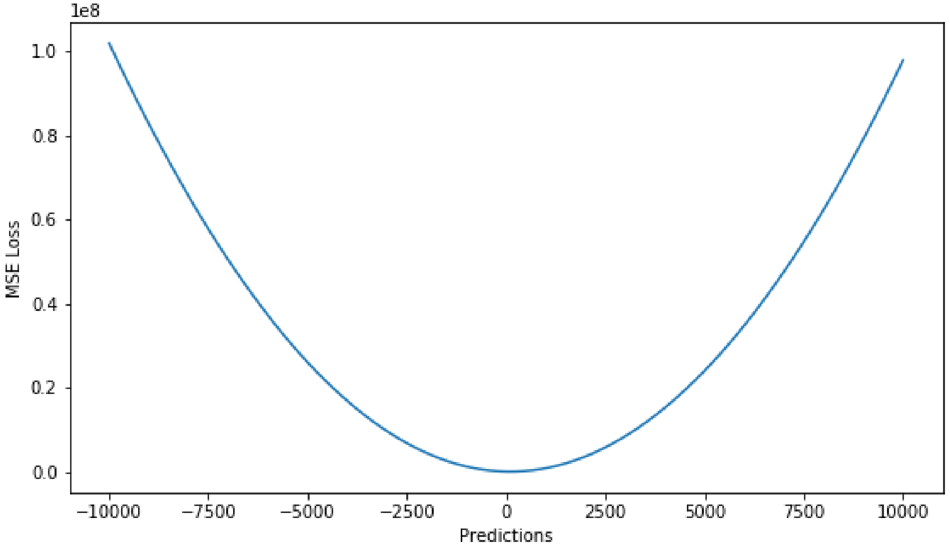
y에 근사하기 위해 y와 f(x)의 제곱 평균으로 정의된 함수로,
값이 낮을 수록 실제값과 예측값의 차이가 작다는 걸 의미

$$MSE(y, f) := E[(y - f(x))^2] = \frac{1}{N} \cdot \sum_{i=1}^{N} (y - f(x))^2$$

예측값과 실제값의 차이는 | y - f(x) | 의 형태로도 구할 수 있음
→ 왜 제곱을 할까?

Unit 01 | MSE - MSE의 개념 (2)

MSE 사용의 의미



1. 2차 함수 형태
   - Convex한 형태를 가져 최적화 과정에서 오차가 최소가 되는 지점, 즉 global minimum을 보장
   - 모든 x에 대해 미분 가능하기 때문에 경사하강법과 같은 최적화 알고리즘 적용에 있어 계산 편리
2. 오차에 대한 민감도
   - 예측값과 실제값의 차이를 제곱하기 때문에 큰 오차에 대해서 더 큰 페널티 부여

Unit 01 | MSE - MSE의 개념 (3)

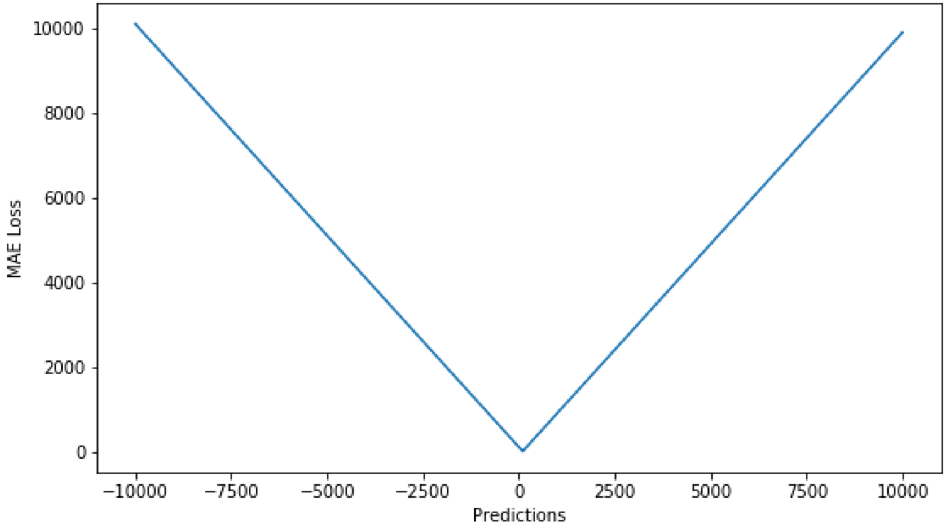
MSE의 오차 민감도

제곱을 적용하기 때문에 작은 값은 더 작아지고 큰 값은 더 커지는 왜곡이 발생하여,
이상치의 영향이 커지는 상황이 발생함

→ 절대적인 성능 지표는 아님. 상황에 따른 지표 선택 필요.

Unit 01 | MSE - MSE vs MAE vs RMSE (1)

MAE (Mean Absolute Error)



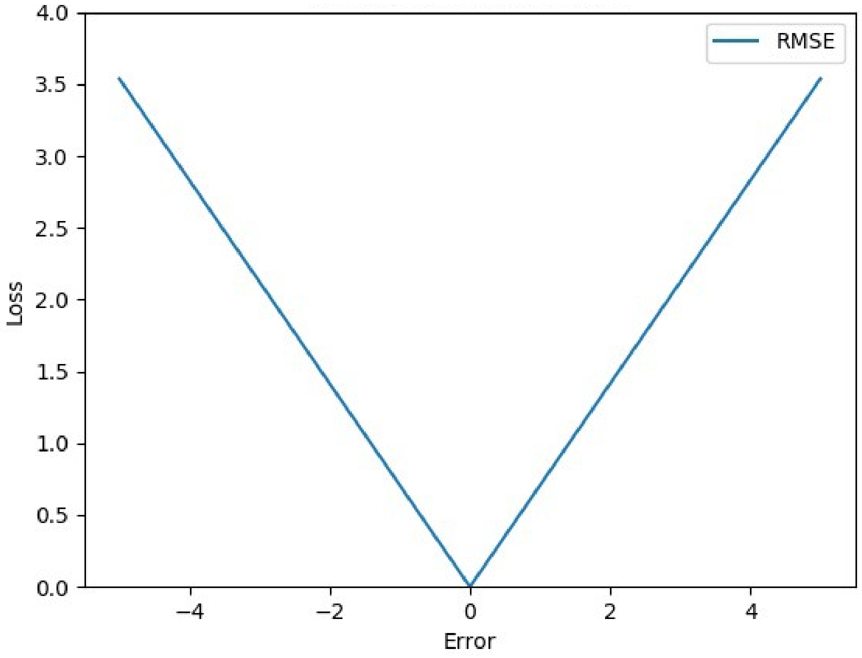
y에 근사하기 위해 y와 f(x)의 절대값 평균으로 정의된 함수로,
절댓값만을 취하기 때문에 오차가 그 자체의 의미만을 가져 이상치에 민감하지 않음
→ 큰 오차에 더 큰 비중을 두는 MSE와의 차이점

- 단점: 절댓값 함수 형태

→ 미분 불가능한 특정 지점에 대한 함수의 기울기(미분값)이 정의되지 않기 때문에 최적화 알고리즘 적용이 어려움

Unit 01 | MSE - MSE vs MAE vs RMSE (2)

RMSE (Root Mean Squared Error)



MSE에 Root를 씌운 형태로, 작은 값이 더 작아지고 큰 값이 더 커지는 MSE의 단점 보완

이상치에 대한 민감도가 MSE보다 적고 MAE보단 크기 때문에 이상치를 적절히 다룬다고 여겨짐

→ 그러나 루트를 취하면서 미분 불가능한 지점 발생한다는 단점이 있음

Unit 01 | MSE - MSE vs MAE vs RMSE (3)

정리

	MAE	MSE	RMSE
오차 민감도	모든 오차를 동등하게 처리	큰 오차에 더 큰 가중치 부여	큰 오차에 더 큰 가중치 부여
이상치 민감도	이상치에 Robust	이상치에 민감	이상치에 민감
단위 및 해석	데이터 단위와 동일하여 직관적 해석	데이터 단위와 달라 상대적으로 직관적이지 않은 해석	데이터 단위와 동일하여 직관적 해석
사용 시점	이상치가 있고, 그 영향을 무시할 경우	이상치의 영향을 반영해야 하는 경우	이상치의 영향을 반영하면서 단위의 해석을 직관적으로 유지하고자 할 때

Unit 01 | MSE - Cross Validation의 개념 (1)

이러한 loss function들은 모델의 성능을 나타내고 parameter update에 사용되나,
단일 검증은 데이터 전체의 분포를 제대로 대표하지 못하고 특정 분할에서의 패턴만 학습할 위험이 존재
→ 과적합 발생

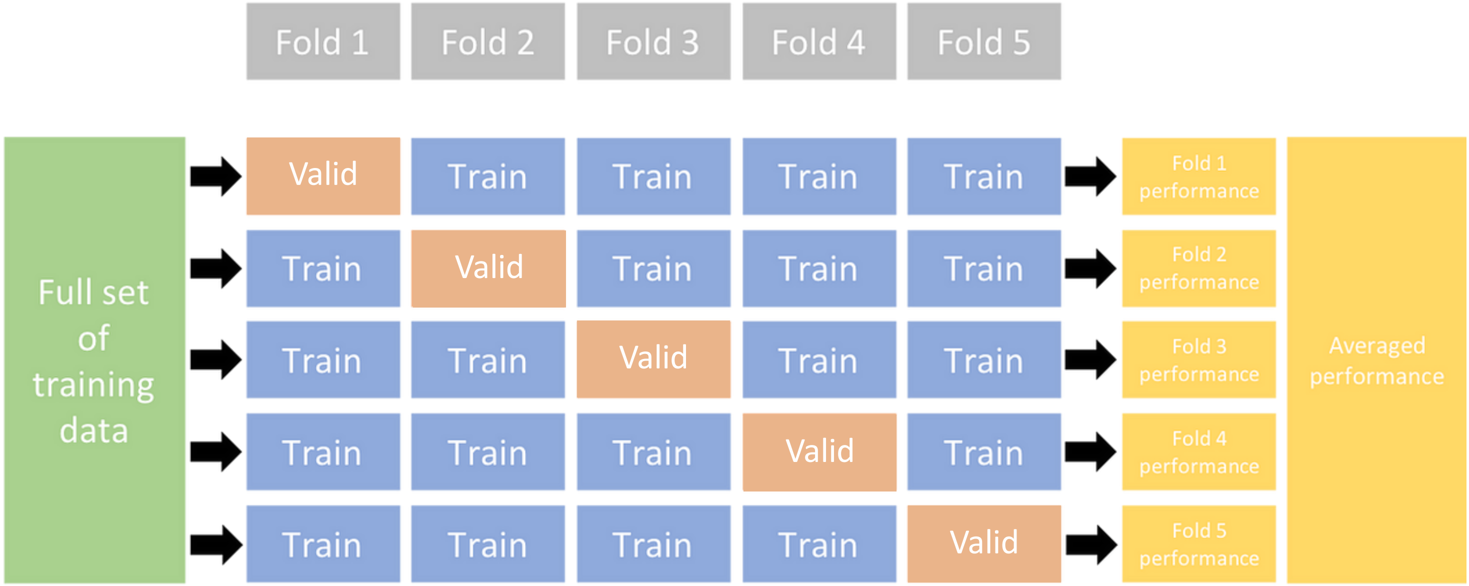
Cross Validation

학습 데이터를 train dataset과 validation dataset으로 나누어 검증하는 과정을 여러번 거치고
최종적으로 test dataset을 통해 모델의 성능을 평가하는 기법

- 장점
  1. 모든 데이터 셋을 평가에 활용하기 때문에 데이터셋이 부족할 때 유용
  2. 여러개의 성능 결과를 통합하여 하나의 결과를 도출해 상대적으로 일반화된 모델 성능 평가 가능

Unit 01 | MSE - Cross Validation의 개념 (2)

K-Fold CV



대표적인 교차 검증 방법으로,
K개의 폴드 세트 중 하나를 validation set, 나머지 K-1개를 train set으로 설정해
학습과 검증을 K번 반복한 뒤 각 검증 평가의 평균으로 최종 평가를 하는 방법

- General K-Fold: 데이터를 K개의 폴드로 분할 (shuffle = True or False)
- Stratified K-Fold: 각 클래스의 분포 비율에 맞게 폴드를 형성함. 불균형 데이터에 사용

Unit 01 | MSE - Cross Validation의 실습 (1)

Training Process

1. 선형 회귀 모델 정의
2. K(폴드 개수 설정)
3. 폴드 생성
   3-1. 인덱스 배열 생성
   3-2. 인덱스 랜덤 정렬
   3-3. 각 폴드 사이즈 설정
4. 교차 검증 수행
   4-1. valid set: i번째 폴드
   4-2. train set: 나머지 폴드
   4-3. 예측 후 mse 계산
   4-4. K번 반복
5. 최종 성능 계산

```
# K값 설정
K = 5

# 폴드 생성 및 교차 검증
indices = np.arange(len(X))
np.random.shuffle(indices)
fold_size = len(X) // K   # fold_size = 20
mse_scores = []
```

```
for i in range(K):
    # index fold: 0 ~ 19, 20 ~ 39, 40 ~ 59, 60 ~ 79, 80 ~ 99
    test_indices = indices[i * fold_size:(i + 1) * fold_size]
    # test index 외 나머지
    train_indices = np.concatenate([indices[:i * fold_size], indices[(i + 1) * fold_size:]])
    X_train, y_train = X[train_indices], y[train_indices]
    X_test, y_test = X[test_indices], y[test_indices]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)
    print(f"Custom Fold {i+1} MSE: {mse:.3f}")

final_mse = np.mean(mse_scores)
```

Unit 01 | MSE - Cross Validation의 실습 (2)

Training Process

1. 선형 회귀 모델 정의
2. K(폴드 개수 설정)
3. 폴드 생성
   3-1. KFold() 적용
4. 교차 검증 수행
   4-1. 각 폴드의 예측 수행
   4-2. mse 계산
5. 최종 성능 계산

```python
# K값 설정
K = 5

# 폴드 생성 및 교차 검증
kf = KFold(n_splits=K, shuffle=True, random_state=42)
mse_scores = []
```

```python
for i, (train_index, test_index) in enumerate(kf.split(X)):     # kf.split(X)
    X_train, y_train = X[train_index], y[train_index]
    X_test, y_test = X[test_index], y[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)
    print(f"Package Fold {i+1} MSE: {mse:.3f}")

final_mse = np.mean(mse_scores)
```

Unit 01 | MSE - Cross Validation의 실습 (실행결과)

```
Custom Fold 1 MSE: 0.008
Custom Fold 2 MSE: 0.012
Custom Fold 3 MSE: 0.014
Custom Fold 4 MSE: 0.013
Custom Fold 5 MSE: 0.020
Custom K-Fold Final MSE: 0.013
No K-Fold MSE: 0.011

Comparison:
Custom K-Fold MSE: 0.013
No K-Fold MSE: 0.011
```

Unit 02 | Training process using MSE

- Process of Training: Target, Estimation, Objective
- Optimization: Handwritten vs Optimization Algorithm

Unit 02 | Training process using MSE

- Process of Training: Target, Estimation, Objective

Summary of training

Target: y
Data: y_i
Estimation: $\hat{y}$
Method: define loss function := MSE

$\Leftrightarrow$ Process:

$$\hat{y} \to y$$

Method:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)
Estimation: $\hat{y}$ = 투빅스 여자 학회원의 **평균**키

$$MSE(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$= \frac{1}{4}\sum_{i=1}^{4}(y_i - \hat{y})^2$$

by solving the optimization, MSE,

$$\hat{y} = \frac{\sum_{i=1}^{N} y_i}{N}$$

Summary of training

Target: y
Data: (x_i, y_i)
Estimation: $\hat{y}$
Method: define loss function := MSE

$\Leftrightarrow$ Objective: $\boxed{x \xrightarrow{?} y}$

x와 y의 상관성은?

Process:

$$\hat{y} \to y$$

where

$$\hat{y}_i = \mathrm{f(x_i)}$$

Target: y = 투빅스 여자 학회원 키
Data: (x_i, y_i) = [(48,158),(53,160), (54,165),(60,168)]
Estimation: $\hat{y}$ = 투빅스 여자 학회원 몸무게의 함수

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$
$$= \frac{1}{4}\sum_{i=1}^{4}(y_i - f(x_i))^2$$
$$= \frac{1}{4}\sum_{i=1}^{4}(y_i - x_i \cdot \beta)^2$$

by solving the optimization, MSE,

$$\hat{y} \Leftrightarrow \hat{\beta} = argmin\ MSE(y, \hat{y})$$

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

HW1

Calculate $\hat{\beta}$

Goal: Find $\hat{y}$ to minimize Loss

$$\hat{y} \Leftrightarrow \hat{\beta} = argmin\ MSE(y, \hat{y})$$

1. Handwritten

$$\frac{\partial\ Loss(y, \widehat{y(\beta)})}{\partial\ \beta} = 0$$

2. Numerical computation
   : Gradient descent ∈ Optimization Algorithms

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\ Loss\left(y, \widehat{y(\beta)}\right)}{\partial\ \beta}\Bigg|_{\beta=\beta^{k-1}}$$

Target: y = 투빅스 여자 학회원 키
Data: (x_i, y_i) = [(48,158),(53,160), (54,165),(60,168)]

$$Loss(y, \hat{y}) \coloneqq MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

1. Hadwritten

$$\hat{\beta} = \boxed{}$$

2. Numerical computation

$$\beta^{(0)} = 0.1$$

$$\beta^{(3)} = \boxed{}$$

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

Pseudo-Code

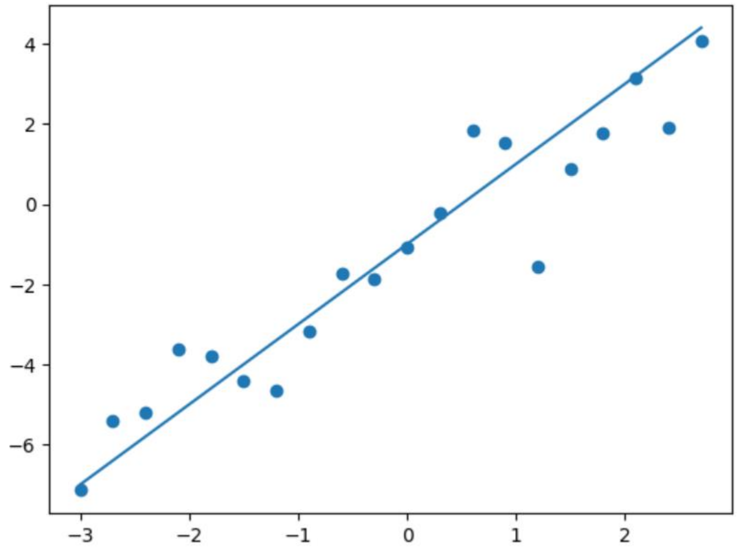
Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



$$\hat{y_i} = 2 * x_i - 1$$

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

Exercise

Training Process

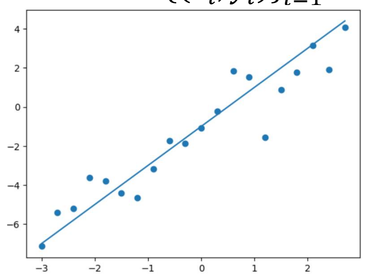
1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

().grad

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



```
def forward(x):
  return x*w+b

def loss_ftn(yhat, y):
  return torch.mean((yhat-y)**2)

torch.manual_seed(2023)
# w,b의 초기점
# w와 b는 param 1개이나, X*w를 할건데 이 차원이 [n,1]이 되도록 w를 [1,1]로 shape을 만들어줌
w = torch.tensor(torch.randn([1,1]), requires_grad=True)
b = torch.tensor(torch.randn([1,1]), requires_grad=True)

print(w,b)


# batch size없이 진행
history = []
lr = 0.1
epochs = 100

for _ in range(epochs):
    yhat =
    loss =

    # 미분
    loss.backward()

    #grad descent
    w.data =
    b.data =

    # 미분값 비워줌.
    w.grad = None
    b.grad = None

    history.append(loss.item())
```

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

Exercise

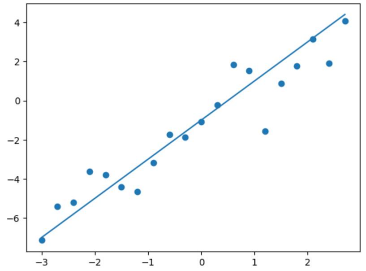
Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



```
def forward(x):
  return x*w+b

def loss_ftn(yhat, y):
  return torch.mean((yhat-y)**2)

torch.manual_seed(2023)
# w,b의 초기점
# w와 b는 param 1개이나, X*w를 할건데 이 차원이 [n,1]이 되도록 w를 [1,1]로 shape을 만들어줌
w = torch.tensor(torch.randn([1,1]), requires_grad=True)
b = torch.tensor(torch.randn([1,1]), requires_grad=True)

print(w,b)


# batch size없이 진행
history = []
lr = 0.1
epochs = 100

for _ in range(epochs):
    yhat = forward(X)
    loss = loss_ftn(yhat, Y)

    # 미분
    loss.backward()

    #grad descent
    w.data = w.data - lr * w.grad
    b.data = b.data - lr * b.grad

    # 미분값 비워줌.
    w.grad = None
    b.grad = None

    history.append(loss.item())
```

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

Exercise

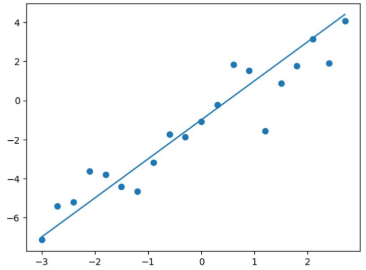
Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



```python
def forward(x):
  return x*w+b
```

torch.nn.linear(k,1, bias = True)

```python
def loss_ftn(yhat, y):
  return torch.mean((yhat-y)**2)

torch.manual_seed(2023)
# w,b의 초기점
# w와 b는 param 1개이나, X*w를 할건데 이 차원이 [n,1]이 되도록 w를 [1,1]로 shape을 만들어줌
w = torch.tensor(torch.randn([1,1]), requires_grad=True)
b = torch.tensor(torch.randn([1,1]), requires_grad=True)

print(w,b)


# batch size없이 진행
history = []
lr = 0.1
epochs = 100

for _ in range(epochs):
    yhat = forward(X)
    loss = loss_ftn(yhat, Y)

    # 미분
    loss.backward()

    #grad descent
    w.data = w.data - lr * w.grad
    b.data = b.data - lr * b.grad

    # 미분값 비워줌.
    w.grad = None
    b.grad = None

    history.append(loss.item())
```

torch.optim.SGD(model.parameters(), lr = 0.1)

Optimizer.step()

Optimizer.zero_grad()

Unit 02 | Training process using MSE

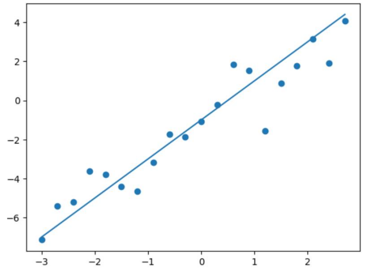
- Optimization: Handwritten vs Optimization Algorithm

Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function
3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



X*w + b0

```python
# model
model = torch.nn.Linear(1,1, bias = True)
my_optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

history=[]
lr=0.1
for epoch in range(100):
    Yhat = model(X)
    loss = loss_ftn(Yhat,Y)

    loss.backward()


    #w.data = w.data-lr*w.grad.data
    #b.data = b.data-lr*b.grad.data
    my_optimizer.step()

    # w.grad.data = None
    # b.grad.data = None
    my_optimizer.zero_grad()

    history.append(loss.item())
```

+ Train with batchsize using DataLoad

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

Exercise

+ Train with batchsize using DataLoad

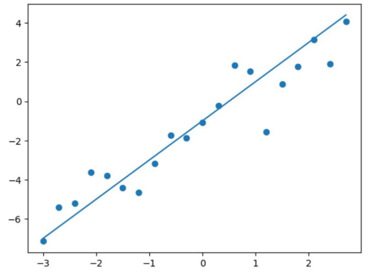
Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



```python
# model
model = torch.nn.Linear(1,1, bias = True)
my_optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

history=[]
lr=0.1
for epoch in range(100):
    Yhat = model(X)
    loss = loss_ftn(Yhat,Y)

    loss.backward()


    #w.data = w.data-lr*w.grad.data
    #b.data = b.data-lr*b.grad.data
    my_optimizer.step()

    # w.grad.data = None
    # b.grad.data = None
    my_optimizer.zero_grad()

    history.append(loss.item())
```

For xx, yy in train_loader:

Unit 02 | Training process using MSE

- Optimization: Handwritten vs Optimization Algorithm

HW2

Fill in the blanks to complete the training

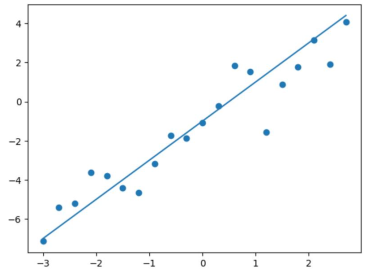
Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$



```python
# model
model = torch.nn.Linear(1,1, bias = True)
my_optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

epochs=100
history=[]
n=len(X)


for epoch in range(epochs):
  LOSS_sum = 0
  for xx, yy in trainloader:
    yhat = model(  )
    loss = loss_ftn(    ,     )
    LOSS_sum = 

    loss.backward()

    my_optimizer.step()
    my_optimizer.zero_grad()

  history.append(LOSS_sum.item()/n)
```

Unit 03 | Training process using MLE

- Def of Likelihood function
- How to find MLE(Maximum Likelihood Estimation)
- Loss function: MSE vs log=likelihood function
- Optimization: Handwritten vs Optimization Algorithm

Unit 03 | Training process using MLE

- Def of Likelihood function

Def

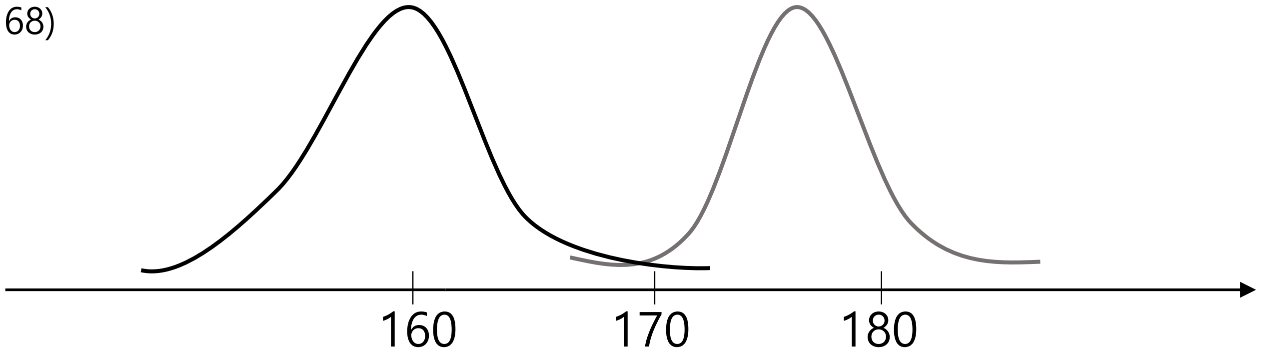
최대 우도 함수(Likelihood function)

: 주어진 데이터에 대해 어떤 **모수**값이 **가장 데이터와 일치할 가능성**을 **최대화**할 수 있는 지를 계산하는 함수

= **가정된 분포**아래 어떤 **모수**값이 주어진 데이터를 잘 뽑아낼 수 있는 지 분포의 **모수**를 찾는 함수

(Example)

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)



$$y \sim N(\mu, \sigma^2)$$

Target: y
Data: y_i
Estimation: $\hat{y}$ = mean of model y
Method: define loss function
:= log-likelihood function

⇔ Process:

$$\hat{y} \rightarrow y$$
$$\hat{y} = E[Y] = \mu$$

Method:

$$L(\mu|y) \rightarrow L(\theta|y)$$

Unit 03 | Training process using MLE

- How to find MLE(Maximum Likelihood Estimation)

Def

최대 우도 함수(Likelihood function)

: 주어진 데이터에 대해 어떤 **모수**값이 **가장 데이터와 일치할 가능성**을 **최대화**할 수 있는 지를 계산하는 함수

= **가정된 분포**아래 어떤 **모수**값이 주어진 데이터를 잘 뽑아낼 수 있는 지 분포의 **모수**를 찾는 함수

※ 데이터의 일치 가능도를 키움
= 그 데이터에 대한 확률 밀도 값을 최대화

$$L(\mu|y) := \prod f_Y(y|\mu)$$

$$y \sim N(\mu, \sigma^2)$$

$$f_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)$$

$y \sim N(\mu, \sigma^2)$

Target: y
Data: y_i
Estimation: $\hat{y}$ = mean of model y
Method: define loss function
:= log-likelihood function

$\Leftrightarrow$ Process:

$\hat{y} \rightarrow y$
$\hat{y} = E[Y] = \mu$

Method:

$L(\mu|y) \rightarrow L(\theta|y)$

Example

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)
Estimation: $\hat{y}$ = mean of model y = $\mu \Leftrightarrow \theta$

(Def)

최대 우도 함수(Likelihood function)

: 주어진 데이터에 대해 어떤 **모수**값이 **가장 데이터와 일치할 가능성**을 **최대화**할 수 있는 지를 계산하는 함수

= **가정된 분포**아래 어떤 **모수**값이 주어진 데이터를 잘 뽑아낼 수 있는 지 분포의 **모수**를 찾는 함수

Method:

$$L(\mu|y) \coloneqq \prod f_Y(y|\mu) = \prod_{i=1}^{N} \mathrm{f}(y_i|\mu)$$

$$l(\mu) \coloneqq \log L(\mu|y)$$

$$\underset{\text{MLE}}{\hat{\mu}} = argmax\ l(\mu)$$

$$\rightarrow \hat{\mu} = \frac{\sum_{i=1}^{N} y_i}{N}$$

$$y \sim N(\mu, \sigma^2)$$

$$f_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)$$

HW3

Calculate of the following MLE

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)
Estimation: $\hat{y}$ = mean of model y = $\mu \Leftrightarrow \theta$

(Def)

최대 우도 함수(Likelihood function)

: 주어진 데이터에 대해 어떤 **모수**값이 **가장 데이터와 일치할 가능성**을 **최대화**할 수 있는 지를 계산하는 함수

= **가정된 분포**아래 어떤 **모수**값이 주어진 데이터를 잘 뽑아낼 수 있는 지 분포의 **모수**를 찾는 함수

Method:

$$L(\mu|y) \coloneqq \prod f_Y(y|\mu) = \prod \boxed{}$$

$$l(\mu) \coloneqq \log L(\mu|y)$$

$$\hat{\mu} = argmax\ l(\mu)$$

$$\rightarrow \hat{\mu} = \boxed{}$$

$$y \sim Exp(\lambda)$$

$$f_Y(y) = \lambda \cdot \exp(-\lambda \cdot y)$$

Summary

Target: y
Data: y_i
Estimation: $\hat{y}$
Method: define loss function
:= log-likelihood function

$\Leftrightarrow$ Process:

$$\hat{y} \to y$$

Method:

$$\max \prod_{i=1}^{N} f(y_i|\mu)$$

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)
Estimation: $\hat{y}$ = 투빅스 여자 학회원의 **분포평균**

$$L(\mu|\,y) = \prod_{i=1}^{N} f(y|\mu)$$

$$y \sim N(\mu, \sigma^2)$$

by maximizing the likelihood,

$$\hat{y} = \hat{\mu} = \frac{\sum_{i=1}^{N} y_i}{N}$$

Unit 03 | Training process using MLE

- Loss function: MSE vs log-likelihood function

Comparison

Target: y
Data: y_i
Estimation: $\hat{y}$
Process:

$$\hat{y} \rightarrow y$$

Method: define loss function

1. MSE

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

2. log-likelihood function

$$\max \prod_{i=1}^{N} \mathrm{f}(y_i|\mu)$$

- Requirement :
Model Assumption

- Objective :
Parameter of distribution

Summary of training

Returning to the process of using x to predict y,

Target: y
Data: (x_i, y_i)
Estimation: $\hat{y}$
Method: define loss function
:=log-likelihood function

$\Leftrightarrow$ Objective: $\boxed{x \xrightarrow{?} y}$

x와 y를 예측하려면?

Process:

$$\hat{y} \to y$$

where

$$\mu_i \Leftrightarrow \hat{y}_i = \mathrm{f(x_i)} = \mathrm{E[y|x_i]} := \mathrm{x}_i \cdot \beta$$

Target: y = 투빅스 여자 학회원 키
Data: (x_i, y_i) = [(48,158),(53,160), (54,165),(60,168)]
Estimation: $\hat{y}$ = 투빅스 여자 학회원 몸무게의 함수

$$\hat{y} = argmax\ l(\mu)$$
$$= argmax\ \log[\prod_{i=1}^{N} \mathrm{f}(y_i|\mu)]$$

$$y_i|x_i \sim N(\mu_i, \sigma^2)$$

$$\hat{y} = argmax\ \log[\prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(y_i - \mu_i)^2}{2 \cdot \sigma^2})]$$
$$= argmax\ \log[\prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(y_i - \mathrm{x}_i \cdot \beta)^2}{2 \cdot \sigma^2})]$$

by estimating of MLE,

$$\hat{y} \Leftrightarrow \hat{\beta} = -argmin\ \log[L(\mu|y)]$$

HW4

Calculate Estimation($\hat{y}$) using the following eqn

Target: y
Data: (x_i, y_i)
Estimation: $\hat{y}$
Method: define loss function
:= log-likelihood ftn

$$\mu_i \Leftrightarrow \quad \hat{y}_i = \mathrm{f}(\mathrm{x_i}) = \mathrm{x}_i \cdot \beta$$

$$y_i \sim N(\mu_i, \sigma^2)$$

$$\hat{y}_i = argmax \log[\prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(y_i - \mathrm{x}_i \cdot \beta)^2}{2 \cdot \sigma^2})]$$

$$\Leftrightarrow -argmin\,[-(y_i - x_i\beta)^2]$$

보통 손으로 계산되지 않아,
수치 계산해야함.
i.e, optim ftn이용 ex SGD

!) y가 normal 분포 가정의 경우,
MSE의 handwritten estimation과 같음!

$\hat{y}_i = \boxed{}$ $\quad \beta = \boxed{}$

Unit 03 | Training process using MLE

- Optimization: Handwritten vs Optimization Algorithm

HW4

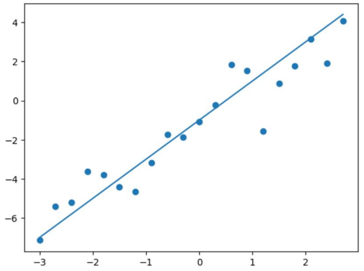
Fill in the blanks to complete the training.

Training Process

1. Data
2. Define yhat = $X \cdot \beta + \beta_0$
   Define loss function
   = log-likelihood function
3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Data: $\{(x_i, y_i)\}_{i=1}^{20}$




```python
'''
sigma = torch.tensor(10.0)
def loss_ftn(yhat, y):
    return


model = torch.nn.Linear(1,1, bias = True)
my_optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

epochs=1000
history=[]
n=len(X)


for epoch in range(epochs):
  LOSS_sum = 0
  for xx, yy in trainloader:
    yhat =
    loss =
    LOSS_sum =

    # weight 미분값 계산
    loss

    # weight update
    my_optimizer.

    # weight grad 초기화
    my_optimizer.

  history.append(LOSS_sum.item()/n)

plt.plot(history)
'''
```

Summary

Target: y
Data: y_i
Estimation: $\hat{y}$
Process:

$$\hat{y} \rightarrow y$$

Method: define loss function

1. MSE

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

2. log-likelihood function

$$\max \prod_{i=1}^{N} \mathrm{f}(y_i|\mu)$$

Training Process

1. Data: (xi,yi)
2. Define yhat = $\mathrm{f}(\mathrm{X}) \coloneqq X \cdot \beta$
   Define loss function
   := MSE, log-likelihood function

   $y \sim N(\mu, \sigma^2)$
3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}|_{\beta=\beta^{k-1}}$$

Further Study

Target: $\boxed{y}$ continuous
Data: y_i
Estimation: $\hat{y}$
Process:

$$\hat{y} \rightarrow y$$

Method: define loss function

1. MSE

$$\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \widehat{y(x)}\right)^2$$

2. log-likelihood function

$$\max \prod_{i=1}^{N} \mathrm{f}(y_i|\mu_i)$$

where $\mu_i = \hat{y}_i = f(x_i)$

Training Process

1. Data: (xi,yi)
2. Define yhat = f(X) := $\boxed{X \cdot \beta}$
   Define loss function
   := MSE, log-likelihood function

   $y \sim \boxed{N(\mu, \sigma^2)}$

3. Initialize parameters
4. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss\left(y, \widehat{y(\beta)}\right)}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

- Futher study: logistic regression

Futher,

Target: $\boxed{y}$ binary
Data: y_i
Estimation: $\hat{y}$
Process:

$$\hat{y} \rightarrow y$$

Method: define loss function

1. MSE

$$\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \widehat{y(x)}\right)^2$$

2. log-likelihood function

$$\max \log[\prod_{i=1}^{N} f(y_i|\mu_i)]$$

where $\mu_i = \hat{y_i} = f(x_i)$

Binary cross entropy

Training Process

1. Data: (xi,yi)
2. Define yhat = $\boxed{f(X) := \frac{1}{1+\exp(-X\beta)}}$ $y \sim \boxed{Ber(p)}$
3. Define loss function
   := MSE, log-likelihood function
4. Initialize parameters
5. Training
   while loss ~= 0:

$$\beta^k = \beta^{k-1} - lr \cdot \frac{\partial\, Loss(y, \widehat{y(\beta)})}{\partial\, \beta}\Big|_{\beta=\beta^{k-1}}$$

Appendix

Appendix 01 | How to calculate MLE in R

- MLE의 수치적 계산 in R

(Example)

Target: y = 투빅스 여자 학회원 키
Data: y_i = (158,160, 165,168)
Estimation: $\hat{y}$ = mean of model y = $\mu \Leftrightarrow \theta$

(Def)

최대 우도 함수(Likelihood function)

: 주어진 데이터에 대해 어떤 **모수**값이 **가장 데이터와 일치할 가능성**을 **최대화**할 수 있는 지를 계산하는 함수

= **가정된 분포**아래 어떤 **모수**값이 주어진 데이터를 잘 뽑아낼 수 있는 지 분포의 **모수**를 찾는 함수

Method:

dnorm(y, mu, sigma)

$$L(\theta|y) \coloneqq \prod f_Y(y|\theta) = \boxed{\prod \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)}$$

$$l(\theta) \coloneqq \log L(\theta|y)$$

$$\boxed{\hat{\theta} = argmax\, l(\theta)}$$

Optimize -sum(dnorm(y, mu, sigma))

$$y \sim N(\mu, \sigma^2)$$

$$f_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)$$

Appendix 01 | How to calculate MLE in R

- MLE의 수치적 계산 in R

(Example)

```
# Example 1: Poisson regression
# Number of military coup in Africa
africa
africa=na.omit(africa)
head(africa)
dim(africa)

y = africa$miltcoup
party = matrix(africa$parties, ncol = 1)
ones = matrix(1, ncol = 1, nrow = length(y))
X = cbind(ones, party)
x = africa$parties
```

```
n.log.lik <-function(y, inputs, betas ){

  betas = matrix(betas, ncol = 1, byrow = FALSE) # [1+p,1] where p = 1
  lambdas = as.vector(X %*% betas) # x:[n, p+1], betas : [n,1], labmda: co
  ret  = - mean(dnorm(y, lambda = lambdas , log = TRUE))
  return (ret)
}

optimization = optim(n.log.lik, y=y, inputs=x, par=c(0,0), method="BFGS")
optimization
beta.hat = optimization$par

yhat = beta.hat[1]+beta.hat[2]*x

mean((y-yhat)^2)
```

Target: y = # of military coup in africa
Data: y_i = (5, 7,...)
Estimation: $\hat{y}$ = mean of model y = $\mu \Leftrightarrow \theta$

$$y \sim N(\mu, \sigma^2)$$

$$f_Y(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)$$

Method:

dnorm(y, mu, sigma)

$$L(\theta|y) \coloneqq \prod f_Y(y|\theta) = \boxed{\prod \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\mu)^2}{2 \cdot \sigma^2}\right)}$$

$$l(\theta) \coloneqq \log L(\theta|y)$$

$$\boxed{\hat{\theta} = argmax\ l(\theta)}$$

Optimize -sum(dnorm(y, mu, sigma))

Q & A

들어주셔서 감사합니다.