lasso algorithm

Jieun Shin

2022-07-26

1. linear regression

1-1. linear regression을 coordinate descent로 풀기

- 목표는 다음과 같다.

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\boldsymbol{\beta}} \frac{1}{2} ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2$$

- gradient는 다음과 같다.

$$0 = \nabla_j f(\boldsymbol{\beta}) = \boldsymbol{X}_j^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) = \boldsymbol{X}_j^T(\boldsymbol{X}_j\boldsymbol{\beta}_j + \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j} - \boldsymbol{y})$$

* j 번째 계수값은 다음과 같다.

$$\boldsymbol{\beta}_j = \frac{\boldsymbol{X}_j^T(\boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j})}{\boldsymbol{X}_j^T\boldsymbol{X}_j} = \frac{\boldsymbol{X}_j(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})}{\boldsymbol{X}_j^T\boldsymbol{X}_j} + \frac{\boldsymbol{X}_j^T\boldsymbol{X}_j\beta_j}{\boldsymbol{X}_j^T\boldsymbol{X}_j} = \frac{\boldsymbol{X}_j^T\boldsymbol{r}}{\boldsymbol{X}_j^T\boldsymbol{X}_j} + \boldsymbol{\beta}_j.$$

여기서 $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$ 이다.

- 알고리즘은 다음을 따른다.

1. 수렴할 때까지 $j = 1, 2, \ldots, p, 1, 2, \ldots, p, \ldots$ 를 반복한다.
2. j 번째 계수값을 다음과 같이 업데이트한다.

$$\boldsymbol{\beta}_j = \frac{\boldsymbol{X}_j^T\boldsymbol{r}}{\boldsymbol{X}_j^T\boldsymbol{X}_j} + \boldsymbol{\beta}_j$$

- 만약 절편항이 포함된 모형일 경우 다음과 같이 업데이트한다.

1. 수렴할 때까지 $j = 1, 2, \ldots, p, 1, 2, \ldots, p, \ldots$ 를 반복한다.
2. β_0 와 β_j 를 다음과 같이 업데이트한다.

$$\beta_0 = \bar{y} - \bar{\boldsymbol{X}}\boldsymbol{\beta},$$
$$\boldsymbol{\beta}_j = \frac{\boldsymbol{X}_j^T(\boldsymbol{y} - \beta_0 - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j})}{\boldsymbol{X}_j^T\boldsymbol{X}_j} = \frac{\boldsymbol{X}_j(\boldsymbol{y} - \beta_0 - \boldsymbol{X}\boldsymbol{\beta})}{\boldsymbol{X}_j^T\boldsymbol{X}_j} + \frac{\boldsymbol{X}_j^T\boldsymbol{X}_j\beta_j}{\boldsymbol{X}_j^T\boldsymbol{X}_j} = \frac{\boldsymbol{X}_j^T\boldsymbol{r}}{\boldsymbol{X}_j^T\boldsymbol{X}_j} + \boldsymbol{\beta}_j.$$

여기서 $\boldsymbol{r} = \boldsymbol{y} - \beta_0 - \boldsymbol{X}\boldsymbol{\beta}$ 이다.

- 코드

```
# 데이터 정리
df = mtcars
y = df[,1]
X = scale(df[,-1])
n = nrow(X); p = ncol(X)
X = as.matrix(X, nrow = n, ncol = p)
attributes(X) = NULL
X = matrix(X, nrow = n, ncol = p)

# 초기값 설정
beta = runif(p, 5, 10) %>% t() %>% t()
m = apply(X, 2, mean)

new_loss = 5
old_loss = 10
t = 0
while(abs(new_loss - old_loss) > 0.001){
  t = t+1

  # j=1
  for(j in 1:p){
    # print(j)
    b0 = mean(y) - m %*% beta
    beta_loss = beta

    r = y - rep(b0,n) - X %*% beta
    beta_loss[j,1] = c(X[,j] %*% r) / sum(X[,j]^2) + beta[j,1]

    new_loss = sum( (y - rep(b0,n) - X %*% beta_loss)^2 )
    old_loss = sum( (y - rep(b0,n) - X %*% beta)^2 )
    # if( abs(new_loss - old_loss) <= 0.001 ){
    #   break
    # }

    beta[j,1] = beta_loss[j,1]  # update
  }
}
c(beta)
```

```
##  [1] -0.05253902  1.50499528 -1.46528726  0.45373607 -3.51716319  1.44943691
##  [7]  0.17834508  1.26087297  0.53593181 -0.40995192
```

```r
lm(y~X)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Coefficients:
## (Intercept)           X1           X2           X3           X4           X5
##     20.0906      -0.1990       1.6528      -1.4729       0.4209      -3.6353
##          X6           X7           X8           X9          X10
##      1.4672       0.1602       1.2576       0.4836      -0.3221
```

1-2. linear regression을 gradient descent로 풀기

$$\beta^{+} = \beta + \eta \cdot \boldsymbol{X}^{T}(\boldsymbol{y} - \boldsymbol{X}\beta)$$

```r
# 데이터 정리
df = mtcars
y = df[,1]
X = df[,-1]
n = nrow(X); p = ncol(X)
X = as.matrix(X, nrow = n, ncol = p)
attributes(X) = NULL
X = matrix(X, nrow = n, ncol = p)
X = cbind(1, X)

# 초기값 설정
beta = runif(p+1, 0, 1) %>% t() %>% t()
eta = 0.01
new_loss = 5
old_loss = 10
t = 0
while(abs(new_loss - old_loss) > 0.001){
  t = t+1
  beta_loss = beta + eta * ginv(X) %*% (y - X %*% beta)

  new_loss = sum( (y - X %*% beta_loss)^2 )
  old_loss = sum( (y - X %*% beta)^2 )

  if( abs(new_loss - old_loss) <= 0.01 ) break

  beta = beta_loss  # update
}

c(beta)
```

```
##  [1] 12.29832068 -0.11105218  0.01367073 -0.02125475  0.78693112 -3.71352998
##  [7]  0.82089557  0.31782075  2.51925602  0.65533899 -0.19912790
```

```r
lm(y~X)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Coefficients:
## (Intercept)           X1           X2           X3           X4           X5
##    12.30337           NA     -0.11144      0.01334     -0.02148      0.78711
##          X6           X7           X8           X9          X10          X11
##    -3.71530      0.82104      0.31776      2.52023      0.65541     -0.19942
```

2. Ridge regression을 coordinate descent로 풀기

- 다음과 같은 문제를 고려한다.

$$\min_{\beta} = \frac{1}{2}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 + \frac{\lambda}{2}||\boldsymbol{\beta}||_2^2$$

- j번째 계수에 대한 gradient는 다음과 같다.

$$\nabla_j R(\boldsymbol{\beta}) = -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\beta_j = 0$$

따라서 j번째 계수의 추정값은 다음과 같다.

$$\beta_j^{ridge} = (\lambda + X_j^T X_j)^{-1} X_j^T(\boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j})$$

1. 수렴할 때까지 $j = 1, 2, \ldots, p, 1, 2, \ldots, p, \ldots$를 반복한다.
2. β_0와 β_j를 다음과 같이 업데이트한다.

$$\begin{aligned} \beta_0 &= \bar{y} - \bar{\boldsymbol{X}}\boldsymbol{\beta}, \\ \boldsymbol{\beta}_j &= (\lambda + X_j^T X_j)^{-1} X_j^T(\boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j}) \end{aligned}$$

```r
# 데이터 정리
df = mtcars
y = df[,1]
X = scale(df[,-1])
n = nrow(X); p = ncol(X)
X = as.matrix(X, nrow = n, ncol = p)
attributes(X) = NULL
X = matrix(X, nrow = n, ncol = p)

# 초기값 설정
lam = 0.2
beta = runif(p, 5, 10) %>% t() %>% t()
m = apply(X, 2, mean)

new_loss = 5
old_loss = 10
t = 0
```

```r
while(abs(new_loss - old_loss) > 0.001){
  t = t+1

  # j=1
  for(j in 1:p){
    # print(j)
    b0 = mean(y) - m %*% beta
    beta_loss = beta

    r = y - rep(b0,n) - X %*% beta
    beta_loss[j,1] = ( t(X[,j]) %*% r ) / (lam + t(X[,j]) %*% X[,j])

    new_loss = sum( (y - rep(b0,n) - X %*% beta_loss)^2 ) + lam * sum(beta_loss^2)/2
    old_loss = sum( (y - rep(b0,n) - X %*% beta)^2 ) + lam * sum(beta^2)/2
    # if( abs(new_loss - old_loss) <= 0.001 ){
    #   break
    # }

    beta[j,1] = beta_loss[j,1]  # update
  }
}

b0; beta
```

```
##          [,1]
## [1,] 20.09062

##             [,1]
##  [1,] -0.6573747
##  [2,] -0.7000998
##  [3,] -0.7616875
##  [4,]  0.5868383
##  [5,] -1.0005916
##  [6,]  0.2871249
##  [7,]  0.4249189
##  [8,]  0.6599593
##  [9,]  0.3599629
## [10,] -0.6734966
```

```r
fit1 = glmnet(X, y, alpha = 0)  # fitting ridge regression
coef(fit1, lam)
```

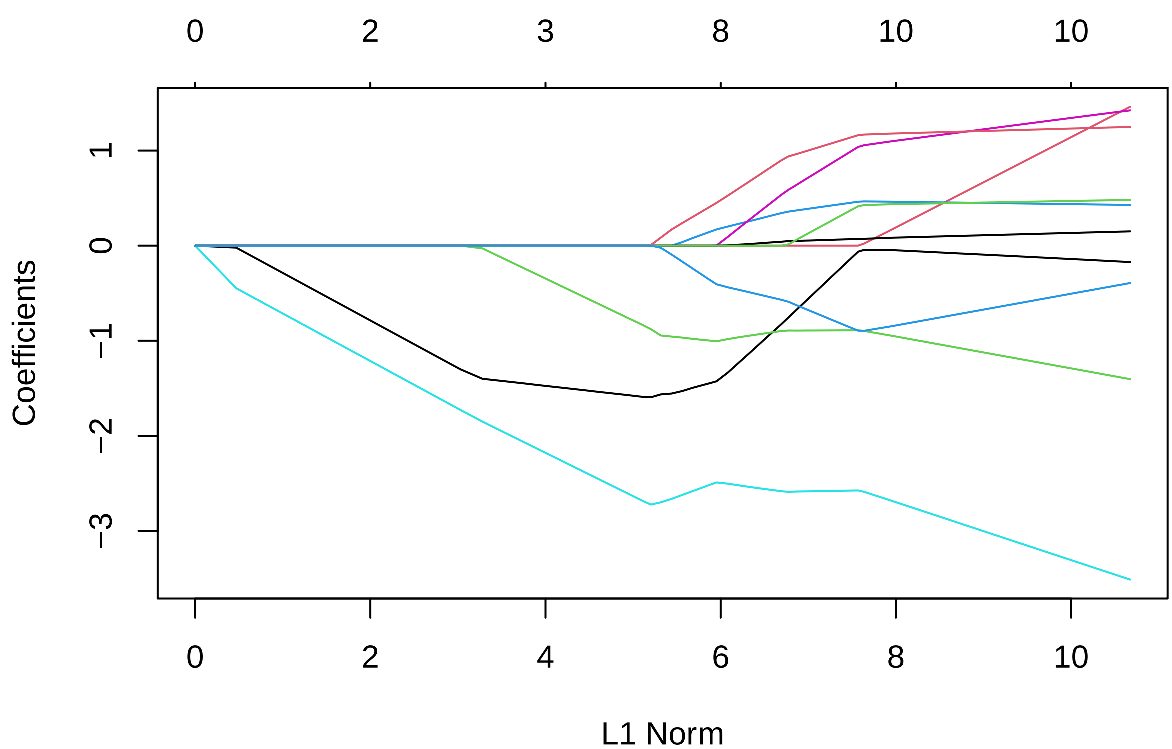
```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept) 20.0906250
## V1          -0.4457943
## V2          -0.2406619
## V3          -0.8959988
## V4           0.5246076
## V5          -1.8514771
## V6           0.5600731
## V7           0.2432525
```

```
## V8          1.0537272
## V9          0.4664281
## V10        -1.0688638
```

3. lasso

3-1. lasso with glm

```
set.seed(100)

df = mtcars
y = df[,1]
X = scale(df[,-1])
n = nrow(X); p = ncol(X)
X = as.matrix(X, nrow = n, ncol = p)
attributes(X) = NULL
X = matrix(X, nrow = n, ncol = p)

fit1 = glmnet(X, y)
plot(fit1)
```



3-1. lasso 알고리즘 구현: coordinate descent

- 다음의 문제를 고려한다:

$$\min_{\boldsymbol{\beta}} = \frac{1}{2}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda$$

- 데이터 $\boldsymbol{X}$ 는 사전에 정규화 시킨다.
- β_j 에 대해 최소화하기 위해 β_j 에 대하여 '미분=0'인 값을 찾는다:

$$\begin{aligned}
0 &= \nabla_j J(\boldsymbol{\beta}) \\
&= -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\partial|\beta_j| \\
&= -X_j^T(\boldsymbol{y} - \boldsymbol{X}_j\boldsymbol{\beta}_j - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j}) + \lambda\partial|\beta_j|
\end{aligned}$$

식을 β_j 에 대하여 전개하면

$$\beta_j = X_j^T(\boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j}) - \lambda\partial|\beta_j|$$

이고, $\boldsymbol{r}_j = \boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j}$ (j 번째 변수에 대한 잔차), $w_j = X_j^T\boldsymbol{r}_j$ (j 번 변수의 OLS)라 두면

$$\beta_j = w_j - \lambda\partial|\beta_j|$$

로 표현된다.

여기서 $|\beta_j|$의 편미분은

$$\partial|\beta_j| = \begin{cases} 1 & \text{if } \beta_j > 0, \\ -1 & \text{if } \beta_j < 0, \\ (-1, 1) & \text{if } \beta_j = 0, \end{cases}$$

이므로 $J(\boldsymbol{\beta})$의 편미분은

$$\nabla_j J(\boldsymbol{\beta}) = \begin{cases} -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) - \lambda & \text{if } \beta_j > 0, \\ -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda & \text{if } \beta_j < 0, \\ (-X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda, -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) - \lambda) & \text{if } \beta_j = 0 \end{cases}$$

로 표현된다.

그러면 $|\beta_j| > 0$ 일 때,

$$-X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \text{ sign}(\beta_j) = 0$$

을 만족하는 β_j 가 lasso의 해가 된다. 이를 찾아보자. 위 식은

$$\beta_j - X_j^T(\boldsymbol{y} - \boldsymbol{X}_{-j}\boldsymbol{\beta}_{-j}) + \lambda \text{ sign}(\beta_j) = 0$$

와 같고 다시 β_j 에 대하여 정리하면

$$\beta_j(c_j) = c_j - \lambda \text{ sign}(\beta_j) = \begin{cases} w_j - \lambda & \text{if } \beta_j > 0, \\ w_j + \lambda & \text{if } \beta_j < 0 \end{cases} = \begin{cases} w_j - \lambda & \text{if } w_j > \lambda, \\ w_j + \lambda & \text{if } w_j < -\lambda \end{cases}$$

이다. 위 값이 lasso의 해이다.

그 다음 $\beta_j = 0$ 일 때,

$$-X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \leq 0 \leq -X_j^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) - \lambda$$

을 만족하는 β_j 가 lasso의 해가 된다. 사실 이미 $\beta_j = 0$ 이라고 정의하였으므로 $\beta_j = 0$ 을 만족하는 범위를 찾는 과정과 같다. 다시 β_j 에 대하여 정리하면

$$
\begin{aligned}
& -X_j^T(\boldsymbol{y} - \boldsymbol{X\beta}) + \lambda \le 0 \le -X_j^T(\boldsymbol{y} - \boldsymbol{X\beta}) - \lambda \\
& \Rightarrow \beta_j - w_j + \lambda \le 0 \le \beta_j - w_j + \lambda \\
& \Rightarrow \lambda \le -\beta_j + w_j \le -\lambda \\
& \Rightarrow -\lambda \le \beta_j - w_j \le \lambda \\
& \Rightarrow |-w_j| \le \lambda \qquad (\because \beta_j = 0) \\
& \Rightarrow |w_j| \le \lambda
\end{aligned}
$$

이므로 $-\lambda \le c_j \le \lambda$ 일 때 $\beta_j(c_j) = 0$ 임을 알 수 있다.

따라서 j 번째 변수에 대한 lasso의 해를 정리하면 다음과 같다.

$$
\hat{\beta}_j^{lasso} = \begin{cases} w_j - \lambda & \text{if } w_j \ge \lambda, \\ 0 & \text{if } -\lambda \le w_j \le \lambda, \\ w_j + \lambda & \text{if } -\lambda \le w_j \end{cases}
$$

lasso의 해를 soft-threshold로 재표현하면 다음과 같다.

$$
\hat{\beta}_j^{lasso} = S(w_j, \lambda) = \text{sign}(w_j)[|w_j| - \lambda]_{(+)}
$$

- 알고리즘

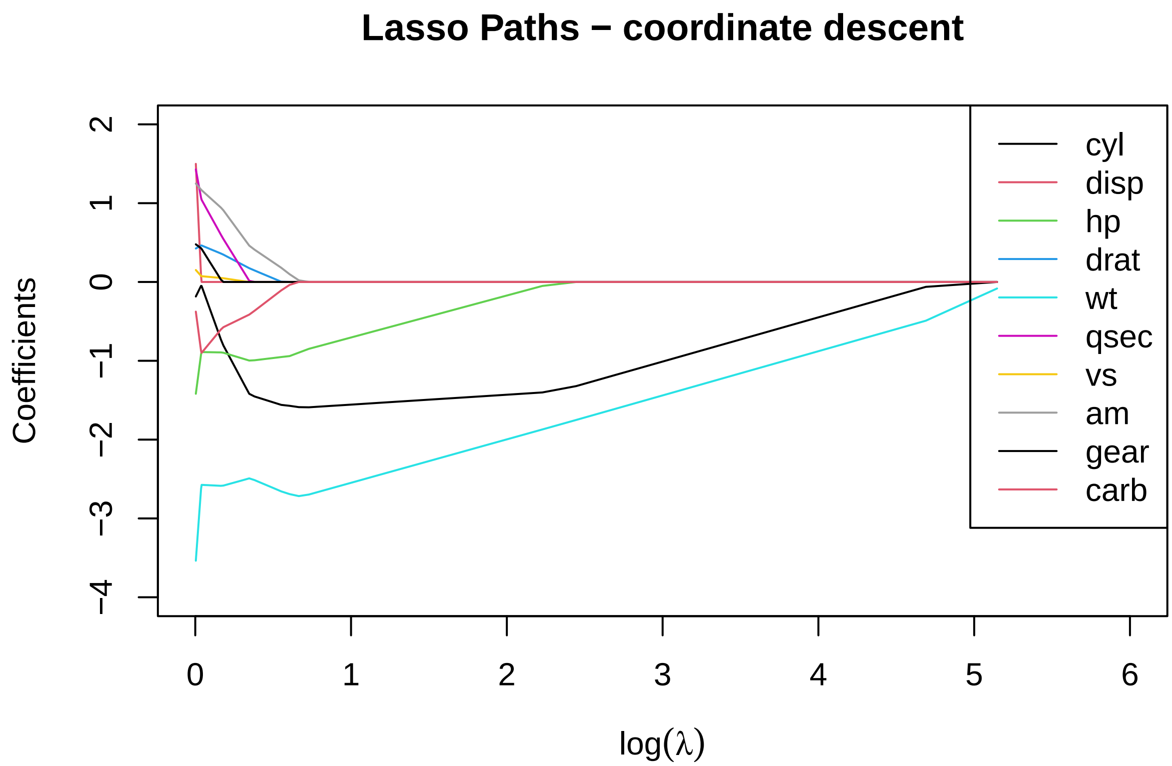
1. 수렴할 때까지 혹은 설정한 최대 반복수까지 $j = 1, 2, \ldots, p, 1, 2, \ldots, p, \ldots$ 를 반복한다.
2. $\beta_0 = \bar{y} - \beta_0 - \bar{\boldsymbol{X}}\boldsymbol{\beta}$ 을 계산한다.
3. $w_j = X_j^T \boldsymbol{r}_j$ 을 계산한다.
4. $\boldsymbol{\beta}_j = S(w_j, \lambda)$ 으로 β_j 를 업데이트한다.

```r
soft_threshold = function(w, lambda){
  if(w < (-lambda) ) {
    return(w + lambda)
  } else if(w > lambda){
    return(w-lambda)
  } else{
    return(0)
  }
}

coordinate_descent_lasso = function(X, y, lambda){

  n = nrow(X); p = ncol(X)
  X = scale(X)   # normalizing
  X = as.matrix(X, nrow = n, ncol = p)
  attributes(X) = NULL
  X = matrix(X, nrow = n, ncol = p)

  # 초기값 설정
  beta = runif(p, -2, 2) %>% t() %>% t()
  m = apply(X, 2, mean)
```

```
  t = 0
  while(t <= 1e+4){
    # while(abs(norm(beta_old) - norm(beta_loss)) > 0.001){
    t = t+1

    # j=1
    for(j in 1:p){
      # print(j)

      y_pred = X %*% beta
      w = t(X[,j]) %*% (y - y_pred + X[,j]*beta[10])
      b0 = mean(y) - m %*% beta

      r = y - rep(b0,n) - X[,-j] %*% beta[-j,1]
      w = c(X[,j] %*% r) / sum(X[,j]^2)
      beta[j,1] = soft_threshold(w, lambda)

    }

  }

  betahat = c(b0, beta)

  return(betahat)
}
```

- Lasso coefficient path

```
set.seed(100)
lambdas = fit1$lambda
lam_len = length(lambdas)

df = mtcars
y = df[,1]
X = df[,-1]

lbeta = matrix(0, nrow = lam_len, ncol = p+1)
for(l in 1:lam_len){
  # print(l)
  lbeta[l,] = coordinate_descent_lasso(X, y, lambdas[l])
}

plot(lambdas, lbeta[,1], xlim = c(0, 6), ylim = c(-4, 2), type = 'n',
     xlab = expression(log(lambda)), ylab = "Coefficients",
     main = "Lasso Paths - coordinate descent")

for(j in 1:p){
  points(lambdas, lbeta[,j+1], type = 'l', col = j)
}

legend("topright", legend = names(X), col = 1:p, lty = 1)
```



3-2. lasso 알고리즘 구현: generalized gradient method (ISTA,)

차후 업데이트 예정