Poisson Regresion and its penalization

Jieun Shin

2023-04-23

데이터는 포아송회귀 가정에 의해 $\log\{\mu_{\boldsymbol{\beta}}(\boldsymbol{x}_i)\} = \beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}_i$에 대하여

$$y_i|\boldsymbol{x}_i \sim \text{Poisson}(\mu_{\boldsymbol{\beta}}(\boldsymbol{x}_i)), \quad i = 1, \cdots, n$$

를 따른다.

독립변수는 $\boldsymbol{x}_i \sim^{iid} N_p(\mathbf{0}, \Sigma)$ 를 따르며, 여기서 $\{\Sigma\}_{ij} = \sigma_{ij} = \rho^{|i-j|}$으로, 실험에서는 $\rho = 0, 0.7$로 지정하였다. 각 회귀계수는 $\beta_0 = 1$ 그리고 $\boldsymbol{\beta} = (1, 1, 1, 0, \cdots, 0)^T \in \mathbb{R}^p, p = 3, 50$으로 둔다. 훈련데이터는 $n = 500$, 시험데이터는 $n_{ts} = 5000$으로 설정하며, 포아송 회귀모형과 RIDGE, LASSO, elastic-net, SCAD 벌점항이 있는 포아송 회귀모형을 비교한다. 벌점화 모형에서 최적의 파라미터는 시험데이터의 deviance를 가장 낮게하는 파라미터로 정하였다. 전 과정을 100번 반복하여 정확도와 변수선택의 성능을 측정한다.

1. 데이터 생성

```r
library(mvtnorm) # 다변량 정규분포 난수생성 함수를 위한 패키지

sim_data = function(train_N = 500, test_N = 5000, p, rho){
  N = train_N + test_N
  if(p == 3){
    beta = rep(1, 3)
    cov = matrix(0, p, p)
    for(i in 1:p){
      for(j in 1:p){
        cov[i,j] = rho^{abs(i-j)}
      }
    }
    X = rmvnorm(N, mean = rep(0,p), sigma = cov)
  }

  if(p == 50){
    beta = c(rep(1, 3), rep(0, p-3))
    cov = matrix(0, p, p)
    for(i in 1:p){
      for(j in 1:p){
        cov[i,j] = rho^{abs(i-j)}
      }
    }
    X = rmvnorm(N, mean = rep(0,p), sigma = cov)
  }

  mu = exp(cbind(1,scale(X)) %*% c(1,beta))
  y = rpois(nrow(X), mu)
```

```
  train_id = sample(1:N, train_N)
  train_X = X[train_id,]
  test_X = X[-train_id,]

  train_y = y[train_id]
  test_y = y[-train_id]

  train_mu = mu[train_id,]
  test_mu = mu[-train_id,]

  return(list(train_X = train_X, train_y = train_y, train_mu = train_mu,
              test_X = test_X, test_y = test_y, test_mu = test_mu, beta = beta))
}
```

2. 포아송 회귀모형

포아송 회귀모형의 로그-가능도함수는 다음과 같다:

$$
\begin{aligned}
\ell(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \log f(y_i; \mu_i) \\
&= \sum_{i=1}^{n} [-\mu_i + y_i \log \mu_i - \log y_i!] \\
&= \sum_{i=1}^{n} [-\exp(\boldsymbol{x}_i^T \boldsymbol{\beta}) + y_i \boldsymbol{x}_i^T \boldsymbol{\beta} - \log y_i!]
\end{aligned}
$$

뉴튼-랩슨 방법을 사용하기 위해 로그-가능도함수의 회귀계수에 대한 1차 도함수와 2차 도함수를 구하면 다음과 같다:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \beta_j} &= \sum_{i=1}^{n} x_{ij}(y_i - \mu_i) \\
\frac{\partial \ell^2}{\partial \beta_j \partial \beta_k} &= -\sum_{i=1}^{n} x_{ij} x_{ik} \mu_i, \quad j, k = 1, \cdots, p
\end{aligned}
$$

그러면 위의 값들이 gradient와 Hessian의 원소가 되어 다음 과정을 수렴할때까지 반복하여 추정치를 구할 수 있다:

$$
\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - H^{-1}(\boldsymbol{\beta}^{(t)}) \nabla \ell(\boldsymbol{\beta}^{(t)})
$$

```
fit_poisson_resgression = function(X, y){
  p = ncol(X)
  sX = scale(X)
  # initialize
  bet = runif(p+1)
  desX = cbind(1, sX)

  t = 0
  while(t < 200){
    t = t + 1
    mu = exp(desX %*% bet)

    # 1. 일차도함수 계산
    # dlogL = rep(0, p+1)
    # for(i in 1:(p+1)) dlogL[i] = sum(desX[,i] %*% (y - mu))
```

```
    dlogL = sapply(1:(p+1), function(i) sum(desX[,i] %*% (y - mu)))
    # 2. 이차도함수 계산
    H = matrix(0, p+1, p+1)
    for(i in 1:(p+1)){
      for(j in 1:(p+1)){
        H[i,j] = -sum(mu * (desX[,i] * desX[,j]))
      }
    }

    bet_new = bet - ginv(H) %*% dlogL
    # print(max(abs(bet_new-bet)))
    if(max(abs(bet_new-bet)) < 0.001) break
    bet = bet_new
  } # end while

  return(list(X = X, y = y, beta = as.vector(bet)))
}
```

3. 벌점화 포아송 회귀모형

먼저 elastic-net 벌점항을 가진 포아송 회귀모형을 만들기 위해 벌점항 $P_\alpha(\boldsymbol{\beta}) = \frac{1}{2}(1-\alpha)||\boldsymbol{\beta}||_2^2 + \alpha||\boldsymbol{\beta}||_1 = \sum_{j=1}^{p}[\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|]$ 과 $\lambda > 0$를 추가한다. elastic-net 회귀계수의 추정은 다음 형태를 최소화하는 문제와 같다:

$$\min_{(\beta_0, \boldsymbol{\beta}) \in \mathbb{R}^{p+1}} \left[\frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \boldsymbol{x}_i^T \boldsymbol{\beta})^2 + \lambda P_\alpha(\boldsymbol{\beta}) \right]$$

이를 elastic-net 벌점항이라 하고, 여기서 $\alpha = 0$이면 RIDGE 벌점항, $\alpha = 1$이면 LASSO 벌점항이 된다. 코드에서도 α로 각 벌점항을 나누었으며, elastic-net 벌점항의 경우 $\alpha = 0.5$로 고정하였다. 최적의 λ는 시험 데이터에서의 deviance를 최소로 만드는 λ를 선택하였다. elastic-net의 해는 soft-threshold $S(z, r) = \text{sign}(z)(|z| - r)_+$로, 각 변수 $j = 1, \cdots, p$에 대해 적용하여 수렴할때까지 반복한다:

$$\hat{\beta}_j = \frac{S(\beta_j^{ols}, \lambda\alpha)}{1 + \lambda(1-\alpha)}$$

여기서 β_j^{ols}는 벌점항이 없는 포아송 회귀모형에서의 추정된 회귀계수를 의미한다.

```
fit_elastic_poisson_regression = function(X, y, lambda, alpha){
  p = ncol(X)
  sX = scale(X)
  # initialize
  bet = runif(p+1)
  desX = cbind(1, sX)
  pen_bet = rep(0, p+1)

  t = loss = 0
  while(t < 200){
    t = t + 1
    nopen_bet = fit_poisson_resgression(X, y)$beta %>% as.vector

    pen_bet_new = rep(0, p+1)
    for(j in 1:(p+1)) pen_bet_new[j] = soft_threshold_elasticnet(nopen_bet[j], lambda, alpha)
```

```
    loss_new = sum(dpois(y, lambda = exp(desX %*% pen_bet_new), log = T)) - sum(abs(pen_bet_new))

    if(abs(loss - loss_new)/abs(loss_new) < 0.01) break
    pen_bet = pen_bet_new
    loss = loss_new
  } # end while

  return(as.vector(pen_bet_new))
}

soft_threshold_elasticnet = function(z, lambda, alpha){
  r = lambda*alpha
  if(z > 0 & r < abs(z)){
    thr = (z - r)/(1+lambda*(1-alpha))
  } else if (z < 0 & r < abs(z)){
    thr = (z + r)/(1+lambda*(1-alpha))
  } else if(r > abs(z)){
    thr = 0
  }
  return(thr)
}

find_optimal_lambda = function(train_X, train_y, test_X, test_y, type, lambda, alpha) {

  if(type %in% c("lasso", "ridge", "elast")){
    out = matrix(0, length(lambda), 2)
    dev = c()
    for(l in 1:20){
      # print(out)
      pred_beta = fit_elastic_poisson_regression(train_X, train_y, lambda = lambda[l], alpha)
      dev[l] = 2 * sum(test_y * log(test_y/exp(cbind(1,test_X) %*% pred_beta)+1)
                       - (test_y - exp(cbind(1,test_X) %*% pred_beta)))
      out[l,] = c(lambda[l], mean(dev))
    }
  }
  if (type == "scad"){
    out = matrix(0, length(lambda), 2)
    dev = c()
    for(l in 1:20){
      # print(out)
      pred_beta = fit_scad_poisson_regression(train_X, train_y, lambda = lambda[l], alpha)
      dev[l] = 2 * sum(test_y * log(test_y/exp(cbind(1,test_X) %*% pred_beta)+1)
                       - (test_y - exp(cbind(1,test_X) %*% pred_beta)))
      out[l,] = c(lambda[l], mean(dev))
    }
  }

  colnames(out) = c("lambda", "dev")
  out = as.data.frame(out)

  return(out)
}
```

그 다음 SCAD 벌점항을 가진 포아송 회귀모형을 만들기 위해 다음과 같은 벌점항을 추가한다:

$$P(\boldsymbol{\beta}) = \begin{cases} \lambda|\beta|, & \text{if } |\beta| < \lambda, \\ \frac{2\alpha\lambda|\beta| - |\beta|^2 - \lambda^2}{2(\alpha-1)}, & \text{if } \lambda < |\beta| \leq \alpha\lambda, \\ \frac{\lambda^2(\alpha+1)}{2}, & \text{if } |\beta| \geq \alpha\lambda. \end{cases}$$

여기서 $\lambda > 0, \alpha > 2$이다. 그러면 SCAD 회귀계수의 추정은 다음 형태를 최소화하는 문제와 같다:

$$\min_{(\beta_0, \boldsymbol{\beta}) \in \mathbb{R}^{p+1}} \left[\frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \boldsymbol{x}_i^T \boldsymbol{\beta})^2 + P(\boldsymbol{\beta}) \right]$$

이때 α도 파라미터 서치가 필요하지만, 계산의 단순화를 위해 [Fan and Li, 2001]에서 권장하는 값인 3.7로 고정하였다. 최적의 λ는 시험 데이터에서의 deviance를 최소로 만드는 λ를 선택하였다. SCAD의 해는 다음과 같이 정의되며, 각 회귀계수 $j = 1, \cdots, p$에 대해 적용하여 수렴할때까지 반복한다:

$$\hat{\beta}_j = \begin{cases} S(\beta_j^{ols}, \lambda\alpha), & \text{if } |\beta| < \lambda, \\ \frac{(\alpha-1)\beta_j^{ols} - \text{sign}(\beta_j^{ols})\alpha\lambda}{\alpha-2}, & \text{if } \lambda < |\beta| \leq \alpha\lambda, \\ \beta_j^{ols}, & \text{if } |\beta| \geq \alpha\lambda. \end{cases}$$

```r
fit_scad_poisson_regression = function(X, y, lambda, alpha){
  p = ncol(X)
  sX = scale(X)
  # initialize
  bet = runif(p+1)
  desX = cbind(1, sX)
  pen_bet = rep(0, p+1)

  t = loss = 0
  while(t < 200){
    t = t + 1
    nopen_bet = fit_poisson_resgression(X, y)$beta %>% as.vector

    pen_bet_new = rep(0, p+1)
    for(j in 1:(p+1)) pen_bet_new[j] = soft_threshold_scad(nopen_bet[j], lambda, alpha)

    # pen = c()
    # for(j in 1:(p+1)) pen[j] = scad_penalty(pen_bet_new[j], lambda, alpha)
    loss_new = sum(dpois(y, lambda = exp(desX %*% pen_bet_new), log = T))

    if(abs(loss - loss_new)/abs(loss_new) < 0.01) break
    pen_bet = pen_bet_new
    loss = loss_new
  } # end while

  return(as.vector(pen_bet_new))
}

soft_threshold_scad = function(z, lambda, alpha){
  if(abs(z) <= 2*lambda){
    thr = sign(z) * max(0, z - lambda)
  } else if (2*lambda < abs(z) & abs(z) <= alpha * lambda){
    thr = ((alpha-1)*z - sign(z)*alpha*lambda) / (alpha-2)
  } else {
```

```
    thr = z
  }
  return(thr)
}
```

3. 실험의 평가

실험의 평가는 크게 회귀계수 추정의 정확성과 변수선택의 성능, 그리고 컴퓨팅 시간의 측정으로 한다.

먼저 회귀계수 추정의 정확성은 MSE, Variance, Bias의 세 가지 측도로 평가하였으며, $k = 1, \cdots, N = 100$번의 반복에 대해 각 측정값은 다음과 같이 계산한다:

- MSE: $\frac{1}{N}\sum_{k=1}^{N}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})^T(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}) = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{p}(\hat{\beta}_{jk} - \beta_{jk})^2$
- Variance: $tr\{\frac{1}{N}\sum_{k=1}^{N}(\hat{\boldsymbol{\beta}}_j - \bar{\boldsymbol{\beta}})^T(\hat{\boldsymbol{\beta}}_j - \bar{\boldsymbol{\beta}})\} = \sum_{j=1}^{p}\frac{1}{N}\sum_{k=1}^{N}(\hat{\beta}_{jk} - \bar{\beta}_j)^2$
- Bias: $\mathbf{1}^T|\bar{\boldsymbol{\beta}} - \boldsymbol{\beta}| = \sum_{j=1}^{p}|\bar{\beta}_j - \beta_j|$

그 다음 변수선택의 성능은 CS (옳게 선택된 변수의 갯수), IS (잘못 선택된 변수의 갯수), AC (변수선택 결과가 참과 같으면 1, 그렇지 않으면 0)의 세 가지 측도로 평가하였으며, 컴퓨팅 시간은 `system.time`함수로 측정하였다. 아래 코드는 평가 측도를 계산하기 위해 구현한 코드이다.

```
return_AC = function(true, pois, ridge, lasso, elast, scad){

  pois_NK = do.call("cbind", pois)
  ridge_NK = do.call("cbind", ridge)
  lasso_NK = do.call("cbind", lasso)
  elast_NK = do.call("cbind", elast)
  scad_NK = do.call("cbind", scad)

  pois_NK_CS = ifelse(pois_NK > 0, 1, 0)
  ridge_NK_CS = ifelse(ridge_NK > 0, 1, 0)
  lasso_NK_CS = ifelse(lasso_NK > 0, 1, 0)
  elast_NK_CS = ifelse(elast_NK > 0, 1, 0)
  scad_NK_CS = ifelse(scad_NK > 0, 1, 0)

  p = nrow(pois_NK_CS)
  pois_CS = sapply(1:N_rep, function(j) sum(pois_NK_CS[,j] == true) == p)
  ridge_CS = sapply(1:N_rep, function(j) sum(ridge_NK_CS[,j] == true) == p)
  lasso_CS = sapply(1:N_rep, function(j) sum(lasso_NK_CS[,j] == true) == p)
  elast_CS = sapply(1:N_rep, function(j) sum(elast_NK_CS[,j] == true) == p)
  scad_CS = sapply(1:N_rep, function(j) sum(scad_NK_CS[,j] == true) == p)

  pois_CS = ifelse(pois_CS, 1, 0)
  ridge_CS = ifelse(ridge_CS, 1, 0)
  lasso_CS = ifelse(lasso_CS, 1, 0)
  elast_CS = ifelse(elast_CS, 1, 0)
  scad_CS = ifelse(scad_CS, 1, 0)

  return(list('mean' = c(mean(pois_CS), mean(ridge_CS), mean(lasso_CS), mean(elast_CS), mean(scad_CS)),
              'se' = c(sd(pois_CS), sd(ridge_CS), sd(lasso_CS), sd(elast_CS), sd(scad_CS))/sqrt(N_rep)))
}

return_IS = function(true, pois, ridge, lasso, elast, scad){
  pois_NK = do.call("cbind", pois)
  ridge_NK = do.call("cbind", ridge)
```

```
  lasso_NK = do.call("cbind", lasso)
  elast_NK = do.call("cbind", elast)
  scad_NK = do.call("cbind", scad)

  pois_NK_CS = ifelse(pois_NK > 0, 1, 0)
  ridge_NK_CS = ifelse(ridge_NK > 0, 1, 0)
  lasso_NK_CS = ifelse(lasso_NK > 0, 1, 0)
  elast_NK_CS = ifelse(elast_NK > 0, 1, 0)
  scad_NK_CS = ifelse(scad_NK > 0, 1, 0)

  pois_CS = sapply(1:N_rep, function(j) sum(pois_NK_CS[,j] != true))
  ridge_CS = sapply(1:N_rep, function(j) sum(ridge_NK_CS[,j] != true))
  lasso_CS = sapply(1:N_rep, function(j) sum(lasso_NK_CS[,j] != true))
  elast_CS = sapply(1:N_rep, function(j) sum(elast_NK_CS[,j] != true))
  scad_CS = sapply(1:N_rep, function(j) sum(scad_NK_CS[,j] != true))

  return(list('mean' = c(mean(pois_CS), mean(ridge_CS), mean(lasso_CS), mean(elast_CS), mean(scad_CS)),
              'se' = c(sd(pois_CS), sd(ridge_CS), sd(lasso_CS), sd(elast_CS), sd(scad_CS))/sqrt(N_rep)))
}

return_CS = function(true, pois, ridge, lasso, elast, scad){
  pois_NK = do.call("cbind", pois)
  ridge_NK = do.call("cbind", ridge)
  lasso_NK = do.call("cbind", lasso)
  elast_NK = do.call("cbind", elast)
  scad_NK = do.call("cbind", scad)

  pois_NK_CS = ifelse(pois_NK > 0, 1, 0)
  ridge_NK_CS = ifelse(ridge_NK > 0, 1, 0)
  lasso_NK_CS = ifelse(lasso_NK > 0, 1, 0)
  elast_NK_CS = ifelse(elast_NK > 0, 1, 0)
  scad_NK_CS = ifelse(scad_NK > 0, 1, 0)

  pois_CS = sapply(1:N_rep, function(j) sum(pois_NK_CS[,j] == true))
  ridge_CS = sapply(1:N_rep, function(j) sum(ridge_NK_CS[,j] == true))
  lasso_CS = sapply(1:N_rep, function(j) sum(lasso_NK_CS[,j] == true))
  elast_CS = sapply(1:N_rep, function(j) sum(elast_NK_CS[,j] == true))
  scad_CS = sapply(1:N_rep, function(j) sum(scad_NK_CS[,j] == true))

  return(list('mean' = c(mean(pois_CS), mean(ridge_CS), mean(lasso_CS), mean(elast_CS), mean(scad_CS)),
              'se' = c(sd(pois_CS), sd(ridge_CS), sd(lasso_CS), sd(elast_CS), sd(scad_CS))/sqrt(N_rep)))
}

return_BIAS = function(true, pois, ridge, lasso, elast, scad){
  pois_NK = do.call("cbind", pois)
  pois_bark = rowMeans(pois_NK)

  ridge_NK = do.call("cbind", ridge)
  ridge_bark = rowMeans(ridge_NK)

  lasso_NK = do.call("cbind", lasso)
  lasso_bark = rowMeans(lasso_NK)
```

```r
  elast_NK = do.call("cbind", elast)
  elast_bark = rowMeans(elast_NK)

  scad_NK = do.call("cbind", scad)
  scad_bark = rowMeans(scad_NK)

  pois_BIAS = sum(abs(true - pois_bark))
  ridge_BIAS = sum(abs(true - pois_bark))
  lasso_BIAS = sum(abs(true - pois_bark))
  elast_BIAS = sum(abs(true - pois_bark))
  scad_BIAS = sum(abs(true - pois_bark))

  return(c(pois_BIAS, ridge_BIAS, lasso_BIAS, elast_BIAS, scad_BIAS))
}

return_VAR = function(pois, ridge, lasso, elast, scad){
  pois_NK = do.call("cbind", pois)
  pois_bark = rowMeans(pois_NK)

  ridge_NK = do.call("cbind", ridge)
  ridge_bark = rowMeans(ridge_NK)

  lasso_NK = do.call("cbind", lasso)
  lasso_bark = rowMeans(lasso_NK)

  elast_NK = do.call("cbind", elast)
  elast_bark = rowMeans(elast_NK)

  scad_NK = do.call("cbind", scad)
  scad_bark = rowMeans(scad_NK)

  p = length(pois_bark)
  pois_VAR = sum( sapply(1:p, function(j) {mean((pois_NK[j,] - pois_bark[j])^2)}) )
  ridge_VAR = sum( sapply(1:p, function(j) {mean((ridge_NK[j,] - ridge_bark[j])^2)}) )
  lasso_VAR = sum( sapply(1:p, function(j) {mean((lasso_NK[j,] - lasso_bark[j])^2)}) )
  elast_VAR = sum( sapply(1:p, function(j) {mean((elast_NK[j,] - elast_bark[j])^2)}) )
  scad_VAR = sum( sapply(1:p, function(j) {mean((scad_NK[j,] - scad_bark[j])^2)}) )

  return(c(pois_VAR, ridge_VAR, lasso_VAR, elast_VAR, scad_VAR))
}

return_MSE = function(true, pois, ridge, lasso, elast, scad){
  pois_iter = lapply(1:N_rep, function(j) {sum((pois[[j]] - true)^2)})
  ridge_iter = lapply(1:N_rep, function(j) {sum((ridge[[j]] - true)^2)})
  lasso_iter = lapply(1:N_rep, function(j) {sum((lasso[[j]] - true)^2)})
  elast_iter = lapply(1:N_rep, function(j) {sum((elast[[j]] - true)^2)})
  scad_iter = lapply(1:N_rep, function(j) {sum((scad[[j]] - true)^2)})

  pois_mse = mean(do.call("c", pois_iter))
  ridge_mse = mean(do.call("c", ridge_iter))
  lasso_mse = mean(do.call("c", lasso_iter))
  elast_mse = mean(do.call("c", elast_iter))
  scad_mse = mean(do.call("c", scad_iter))
```

```
  return(c(pois_mse, ridge_mse, lasso_mse, elast_mse, scad_mse))
}
```

4. 실행

위의 코드들은 아래의 코드로 실행할 수 있다.

```
# N_rep = 100  # 반복수
# lambda_grid = seq(0.001, 1, length.out = 20)
#
#
# MSE = VAR = BIAS = CSm = ISm = ACm = CSs = ISs = ACs = time = matrix(0, 4, 5)
#
# k = 0
# for(pp in c(3, 50)){
#   for(rr in c(0, 0.7)){
#     k = k + 1
#     print(k)
#     fit_pois = fit_ridge = fit_lasso = fit_elast = fit_scad = list()
#
#     for(iter in 1:N_rep){
#       set.seed(iter)
#       sim_dat = sim_data(train_N = 500, test_N = 5000, p = 50, rho = 0)  # 데이터 생성
#       true_beta = c(1, sim_dat$beta)
#
#       tr_X = sim_dat$train_X
#       tr_y = sim_dat$train_y
#       te_X = sim_dat$test_X
#       te_y = sim_dat$test_y
#
#       # 1. POIS
#       t1 = system.time({
#         fit_pois[[iter]] = fit_poisson_resgression(tr_X, tr_y)$beta
#       })[3]
#
#       # 2. ridge
#       t2 = system.time({
#         find_ridge_lambda = find_optimal_lambda(tr_X, tr_y, te_X, te_y, type = "ridge", lambda_grid, 
#         opt_lambda = lambda_grid[which.min(find_ridge_lambda[,2])]
#         fit_ridge[[iter]] = fit_elastic_poisson_regression(tr_X, tr_y, lambda = opt_lambda, alpha = 0
#       })[3]
#
#       # 3. lasso
#       t3 = system.time({
#         find_lasso_lambda = find_optimal_lambda(tr_X, tr_y, te_X, te_y, type = "lasso", lambda_grid, 
#         opt_lambda = lambda_grid[which.min(find_lasso_lambda[,2])]
#         fit_lasso[[iter]] = fit_elastic_poisson_regression(tr_X, tr_y, lambda = opt_lambda, alpha = 1
#       })[3]
#
#       # 4. elast
#       t4 = system.time({
#         find_elast_lambda = find_optimal_lambda(tr_X, tr_y, te_X, te_y, type = "elast", lambda_grid, 
#         opt_lambda = lambda_grid[which.min(find_elast_lambda[,2])]
```

```r
#         fit_elast[[iter]] = fit_elastic_poisson_regression(tr_X, tr_y, lambda = opt_lambda, alpha = 0
#       })[3]
#
#       # 5. scad
#       t5 = system.time({
#         find_scad_lambda = find_optimal_lambda(tr_X, tr_y, te_X, te_y, type = "scad", lambda_grid, al
#         opt_lambda = lambda_grid[which.min(find_scad_lambda[,2])]
#         fit_scad[[iter]] = fit_scad_poisson_regression(tr_X, tr_y, lambda = opt_lambda, alpha = 3.7)
#       })[3]
#     }
#
#     MSE[k,] = return_MSE(true_beta, fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#     VAR[k,] = return_VAR(fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#     BIAS[k,] = return_BIAS(true_beta, fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#
#     CS_result = return_CS(true_beta, fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#     IS_result = return_IS(true_beta, fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#     AC_result = return_AC(true_beta, fit_pois, fit_ridge, fit_lasso, fit_elast, fit_scad)
#
#     CSm[k,] = CS_result$mean
#     ISm[k,] = IS_result$mean
#     ACm[k,] = AC_result$mean
#     CSs[k,] = CS_result$se
#     ISs[k,] = IS_result$se
#     ACs[k,] = AC_result$se
#
#     time[k,] = c(t1, t2, t3, t4, t5)
#
#   } # for rho
# } # for p
```


[1] Fan, J., & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96(456), 1348-1360.

[2] Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.