MTH210: Lab 12 Solutions

Cross-validation

The data is randomly split into K roughly equal-sized parts. For any kth split, the rest of the $K-1$ parts make up the training set and the model is fit to the training set. We then estimate the prediction error for each element in the kth part. Repeating this for all $k = 1, 2, \dots, K$ parts, we have an estimate of the prediction error.

Let $\kappa : \{1, \dots, N\} \mapsto \{1, \dots, K\}$ indicates the partition to which each ith observation belongs. Let $\hat{f}^{-\kappa(i)}(x)$ be the fitted function for the $\kappa(i)th$ partition removed. Then, the estimated prediction error is

$$\mathrm{CV}_K(\hat{f}, \gamma) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{n/K} \sum_{i \in k^{\text{th}}\text{split}} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \gamma)) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \gamma)) \,.$$

The chosen model is the one with γ such that

$$\theta_{\text{chosen}} = \arg\min_{\gamma} \left\{ \mathrm{CV}_K(\hat{f}, \gamma) \right\}$$

The final model is $\hat{f}(X, \gamma_{\text{chosen}})$ fit to all the data.

Points:

- For small K, the bias in estimating the true test error is large since each training data is quite different from the given dataset $\mathcal{D}$.
- The computational burden is lesser when K is small.

Usually, for large datasets, 10-fold or 5-fold CV is common. For small datasets, LOOCV is more common.

Now, let's try a few implementations.

1. **Recall the regression model: $Y = X\beta + \epsilon$, where $\epsilon \sim N_n(0, \sigma^2 I_n)$. In this example subsection, we will focus our attention only on ridge regression estimators. Similar cross-validations can be done for bridge regression. Recall the ridge objective function for a given λ is**

   $$Q_\lambda(\beta) = \frac{(y - X\beta)^T (y - X\beta)}{2} + \frac{\lambda}{2} \beta^T \beta \,.$$

   Recall, the ridge estimator of β is

$$\hat{\beta}_\lambda = (X^T X + \lambda I_n)^{-1} X^T y\,.$$

Here, the solution for β depends on the value of λ. We want to choose λ so that prediction error is minimized. We choose the squared error loss function. To choose the best model in this case, we set a vector of $\lambda : \lambda_1, \ldots, \lambda_m$.

Since closed-form solution for the ridge-regression estimator is available, cross-validation steps will be fairly fast. Thus we choose LOOCV. For each λ_i, we will implement LOOCV and estimate the prediction error. Whichever λ minimzies the prediction error, will be the chosen λ.

Consider the `mtcars` dataset in `R`. The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models). The response is the miles per gallon of the 32 cars. Run a `?mtcars`} in `R` to learn more about the dataset.

We will choose

$$\lambda \in \{i \in (-8, 8) : 10^i\},$$

where i will increase in increments of .1. This makes it a large grid of λs.

```
#####################################################
## Choosing lambda in ridge regression
## using LOOCV
## dataset is mtcars
#####################################################
data(mtcars)
y <- mtcars$mpg # response

# adding intercept
X <- cbind(1, as.matrix(mtcars[ ,-1]))

n <- dim(X)[1]
p <- dim(X)[2]

# vector of lambdas
lam.vec <- c(10^(seq(-8, 8, by = .1)))
# Will store CV error in this.
CV.error <- numeric(length = length(lam.vec))
```

Now things are setup, to implement LOOCV.

```
# for each lambda we will do ridge
for(l in 1:length(lam.vec))
{
  track.cv <- 0
  lam <- lam.vec[l]
  for(i in 1:n)
  {
    # Making training data
    X.train <- X[-i,] # removing ith X
    y.train <- y[-i]  #removing ith y

    # fitting model for training data
    beta.train <- solve(t(X.train) %*% X.train + lam*diag(1,p)) %*% t(X.train) %*% y.train
    # test error
    track.cv <- track.cv + (y[i] - X[i,] %*% beta.train)^2
  }
  CV.error[l] <- track.cv/n
}
```

Now, we have calculated the CV errors for all λs. Now we can find the best one:

```
(chosen.lam <- lam.vec[which.min(CV.error)] )
```

```
[1] 5.011872
```

Which the chosen λ, λ_{chosen}, we should refit the model again on the whole data, to get the final β estimates.

```
beta.final <- solve(t(X) %*% X + chosen.lam*diag(p)) %*% t(X) %*% y
```

2. **Repeat the above process for 5-fold and 10-fold cross-validation. The following code will be useful. It splits the index set $\{1, 2, \ldots, n\}$, into K different folds:**

```
permutation <- sample(1:n, replace = FALSE)
K <- 4

# Making a list of indices for each split
test.index <- split(permutation, rep(1:K, length = n, each = n/K))
```

We will repeat the above process now for K-fold cross-validations.

```
for(l in 1:length(lam.vec))
{
  track.cv <- 0
```

```
  lam <- lam.vec[l]
  for(i in 1:K)
  {
    train.ind <- test.index[[i]]
    # Making training data
    X.train <- X[-train.ind,] # removing ith X
    y.train <- y[-train.ind]  #removing ith y

    # fitting model for training data
    beta.train <- solve(t(X.train) %*% X.train + lam*diag(1,p)) %*% t(X.train) %*% y.train
    # test error
    track.cv <- track.cv + sum((y[train.ind] - X[train.ind,] %*% beta.train)^2)
  }
  CV.error[l] <- track.cv/n
}

# Finding the lambda value that gives best CV-error
(chosen.lam <- lam.vec[which.min(CV.error)] )
```

```
[1] 3.162278
```

```
# refitting for the find value of lambda
beta.final <- solve(t(X) %*% X + chosen.lam*diag(p)) %*% t(X) %*% y
```

3. **For the above dataset, what is the best value of α and λ that minimizes test error for Bridge regression?**

   We will use the values of λ we had before. We will also consider the following grid of values of α between $(0, 1)$.

```
lam.vec <- c(10^(seq(-8, 8, by = .1)))
alpha.vec <- seq(1, 2, length = 10)

# CV.error will be a matrix
CV.error <- matrix(0, ncol = length(alpha.vec), nrow = length(lam.vec))
```

   Recall the Bridge Regression estimation function from Lab 8:

```
# y = response
# X = covariate matrix
# lambda = penlaty term
```

```r
# alpha = bridge term
# max.ter = maximum iterations for the MM algorithm. if max iteration has been reached, the func
# tol = tolerance level for when to stop MM
bridgeReg <- function(y.this, X.this, alpha, lambda, max.iter, tol = 1e-3)
{
  # a value larger than tol
  distance <- tol + 1
  iter <- 0
  p <- dim(X)[2]

  # starting from the ridge solution
  current <- qr.solve(t(X.this) %*% X.this + lambda*diag(p)) %*% t(X.this) %*% y.this

  while(distance > tol)
  {
    iter <- iter + 1
    if(iter > max.iter)
    {
      print("Maximum iterations reached")
      break
    }

    # MM steps
    previous <- current
    mjs <- alpha* abs(current)^(alpha - 2)

    ## using qr.solve since that is more stable than solve
    current <- qr.solve(t(X.this)%*%X.this + lambda/alpha * diag(mjs), tol = 1-09) %*% t(X.this)
    distance <- norm(previous - current, "2")
  }
  #returning the last iterate of the
  return(current)
}
```

Now let's repeat the process above for all values of λ and α. We will repeat the cross-validation splits first:

```r
permutation <- sample(1:n, replace = FALSE)
K <- 4

# Making a list of indices for each split
test.index <- split(permutation, rep(1:K, length = n, each = n/K))
```

```r
for(a in 1:length(alpha.vec))
{
  alpha <- alpha.vec[a]
  print(paste("Current alpha is", alpha))
  for(l in 1:length(lam.vec))
  {
    track.cv <- 0
    lam <- lam.vec[l]
    for(i in 1:K)
    {
      train.ind <- test.index[[i]]
      # Making training data
      X.train <- X[-train.ind, ] # removing ith X
      y.train <- y[-train.ind]  #removing ith y

      # fitting model for training data
      # requires lots of iterations
      beta.train <- bridgeReg(y.train, X.train, alpha, lam, max.iter = 1e5)
      # test error
      track.cv <- track.cv + sum((y[train.ind] - X[train.ind, ] %*% beta.train)^2)
    }
    CV.error[l, a] <- track.cv/n
  }
}
```

```
[1] "Current alpha is 1"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.11111111111111"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
```

```
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.22222222222222"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.33333333333333"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.44444444444444"
[1] "Maximum iterations reached"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.55555555555556"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.66666666666667"
[1] "Maximum iterations reached"
[1] "Current alpha is 1.77777777777778"
[1] "Current alpha is 1.88888888888889"
[1] "Current alpha is 2"
```

Now we find the combination of values of α and λ that give the lowest CV error

```
index <- which(CV.error == min(CV.error), arr.ind = TRUE)

(chosen.lam <- lam.vec[index[1]])
```

```
[1] 63095734
```

```
(chosen.alpha <- alpha.vec[index[2]])
```

```
[1] 2
```

```
# refitting for the find value of lambda
# and alpha
```

```
(beta.final <- bridgeReg(y, X, chosen.alpha, chosen.lam, max.iter = 2e5))
```

```
            [,1]
     -2.36726711
cyl   0.35279315
disp  0.01449144
hp   -0.01940354
drat  1.28846769
wt   -4.05039741
qsec  1.32396772
vs   -0.13774729
am    2.70441233
gear  1.13092110
carb -0.24023807
```

4. **Recall the ridge logistic regression model you fit in Assignment 3. Find the optimal value of λ for the `titanic` dataset:**

```
titanic <- read.csv("https://dvats.github.io/assets/titanic.csv")
```

Let us take my solution to Assignment 3 first:

```
f.gradient <- function(y, X, beta, lam)
{
     # converting beta to compatible matrix form
     beta <- matrix(beta, ncol = 1)
     pi.vec <-  1 / (1 + exp(-X%*%beta/2))
     rtn <- colSums(X* as.numeric(y - pi.vec)/2) - lam*beta
     return(rtn)
}

f.hessian <- function(y, X, beta, lam)
   {
     beta <- matrix(beta, ncol = 1)
     W_i <-  exp(X%*%beta/2) / (1 + exp(X%*%beta/2))^2
     W <- diag(as.numeric(W_i))
     rtn <- - t(X) %*% W %*% X/4 - diag(lam, dim(X)[2])
}

MYlogridge <- function(y, X, lam)
{

    p <- dim(X)[2]
```

```
    n <- length(y)

   tol <- 1e-10
   compare <- 100
   iter <- 1
   # starting from the zero-vector
   grad.vec <- c() # will store gradients here
   beta.current <- rep(0, p)
   beta.new <- beta.current
    while(compare > tol)
       {
         iter <- iter + 1  # tracking iterations
         gradient <- f.gradient(y, X, beta.current, lam)
         hessian <- f.hessian(y, X, beta.current, lam)
         beta.new <- beta.current - qr.solve(hessian) %*% gradient
         grad.vec[iter] <- norm(gradient, "2")
         beta.current <- beta.new
         compare <- grad.vec[iter]
    }
    return(beta.current)
}
```

Now we will implement 5-fold cross validation.

```
titanic <- read.csv("https://dvats.github.io/assets/titanic.csv")
head(titanic)
```

```
  Survived X.Intercept. Sexmale Age SibSp Parch    Fare
1        0            1       1  22     1     0  7.2500
2        1            1       0  38     1     0 71.2833
3        1            1       0  26     0     0  7.9250
4        1            1       0  35     1     0 53.1000
5        0            1       1  35     0     0  8.0500
6        0            1       1  54     0     0 51.8625
```

```
y <- titanic$Survived
X <- as.matrix(titanic[, -1])

n <- length(y)
p <- dim(X)[2]
# everything but the first column is the X

permutation <- sample(1:n, replace = FALSE)
```

```r
K <- 5

# Making a list of indices for each split
test.index <- split(permutation, rep(1:K, length = n, each = n/K))
```

We have made our splits, now we can implement the CV. Recall that the loss function here is the 0-1 loss, or the misclassification loss.

```r
lam.vec <- c(10^(seq(-8, 8, by = .1)))
# Will store CV error in this.
CV.error <- numeric(length = length(lam.vec))

for(l in 1:length(lam.vec))
{
  track.cv <- 0
  lam <- lam.vec[l]
  for(i in 1:K)
  {
    train.ind <- test.index[[i]]
    # Making training data
    X.train <- X[-train.ind,] # removing ith X
    y.train <- y[-train.ind]  #removing ith y

    # fitting model for training data
    beta.train <- MYlogridge(y.train, X.train, lam = lam)

    # estimated probability
    pi <- X[train.ind,] %*% beta.train
    est.yi <- (pi > .5)

    # misclassification error
    track.cv <- track.cv + sum(y[train.ind] != est.yi)
  }
  CV.error[l] <- track.cv/n
}

# Finding the lambda value that gives best CV-error
(chosen.lam <- lam.vec[which.min(CV.error)] )
```

```
[1] 0.6309573
```

```r
# refitting for the find value of lambda
(beta.final <- MYlogridge(y, X, lam = chosen.lam) )
```

```
                    [,1]
X.Intercept.  2.32484673
Sexmale      -4.42916258
Age          -0.03431445
SibSp        -0.68794387
Parch        -0.33838695
Fare          0.03358326
```