CW3

Wen Hans Tan and Eirshad Fahim

2025-02-30

(i) **(2 marks)** Write code to read in the dataset *Salary_Data.csv* and perform a train-validation-test split.

```
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```
library(readr)
Salary_Data <- read_csv("C:/Users/tanwe/OneDrive/Documents/Stats_Machine_Learning/CW3/Salary_Data.csv")
```

```
## Rows: 6699 Columns: 209

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (209): Age, GenderFemale, GenderMale, GenderOther, Education.LevelBachel...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
any(is.na(Salary_Data))
```

```
## [1] FALSE
```

```
# Get shuffled indices
indices <- sample(1:nrow(Salary_Data), nrow(Salary_Data))

# Define split sizes
train_size <- floor(0.5 * nrow(Salary_Data))
val_size <- floor(0.25 * nrow(Salary_Data))
test_size <- nrow(Salary_Data) - train_size - val_size

# Assign data based on exact indices
train_Data <- Salary_Data[indices[1:train_size], ]
valid_Data <- Salary_Data[indices[(train_size+1):(train_size+val_size)], ]
test_Data <- Salary_Data[indices[(train_size+val_size+1):nrow(Salary_Data)], ]

#Confirm Training Validation and Test Size
cat("Training size:", nrow(train_Data), "\n")
```

```
## Training size: 3349
```

```r
cat("Validation size:", nrow(valid_Data), "\n")
```

```
## Validation size: 1674
```

```r
cat("Test size:", nrow(test_Data), "\n")
```

```
## Test size: 1676
```

(ii) **(3 marks)**

```r
my_loss <- function(lambda, train_data, valid_data) {
  #Split X and y from training and validation data
  X_train <- as.matrix(train_data[, !names(train_data) %in% "Salary"])
  y_train <- train_data$Salary
  X_valid <- as.matrix(valid_data[, !names(valid_data) %in% "Salary"])
  y_valid <- valid_data$Salary

  #Fit LASSO model using glmnet
  lasso_model <- glmnet(X_train, y_train, alpha=1,lambda = lambda)

  #Make Predictions using validation data
  y_pred <- predict(lasso_model, s=lambda, newx = X_valid)

  #Compute MSE on validation set
  mse <- mean ((y_valid - y_pred)^2)

  return(mse)
}
```

(iii) **(2 marks)**

```r
sample_lambdas <- function(N) {
  #Generating N random samples from Uniform distribution [0,1]
  lambda <- runif(N, min=0, max=1)

  return(lambda)
}
```

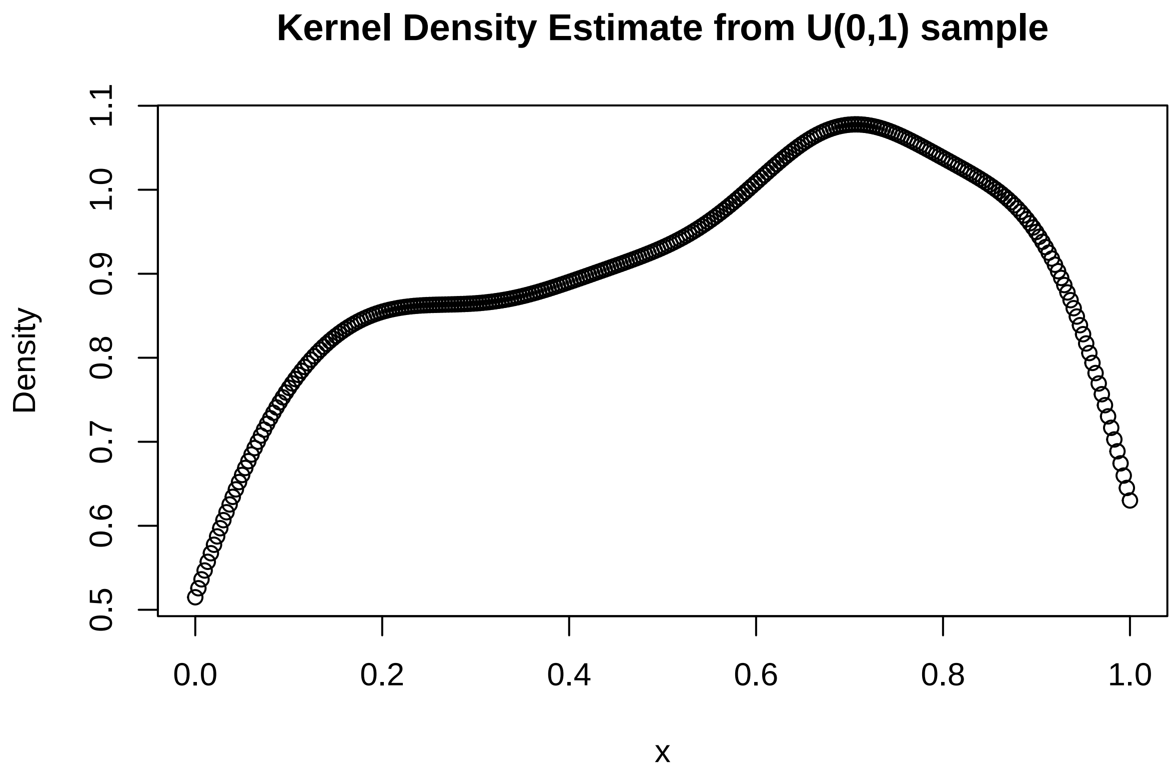
(iv) **(3 marks)**

```r
my_kde <- function(z_values, x_values) {
  #Bandwidth
  h <- 0.1
  #Number of observed data points
  n <- length(z_values)

  #Define Gaussian Kernel function
  gaussian_kernel <- function(x) {
```

```r
    return((1 / sqrt(2 * pi)) * exp(-0.5 * x^2))
  }

  # Compute KDE estimates at each observable x in x_values
  kde_estimates <- sapply(x_values, function(x) {
    kernel_values <- gaussian_kernel((x - z_values) / h)  # Compute kernel values
    return(sum(kernel_values) / (n * h))  # Compute density estimate
  })

  #Array of all kde_estimates at each given x
  return(kde_estimates)
}
```

(v) **(2 marks)**

```r
#Set seed for reproducibality
set.seed(42)

#Observed values to construct the KDE
z_values <- runif(100, 0, 1)

#X-values where KDE is evaluated
x_values <- seq(0, 1, length.out=300)

#Convert z_values and x_values as arrays
z_array <- array(z_values)
x_array <- array(x_values)

#Compute kde estimates
kde_estimates <- my_kde(z_array, x_array)

#Plot KDE
plot(x_values, kde_estimates, main = "Kernel Density Estimate from U(0,1) sample", xlab="x", ylab="Densi
```



(vi) **(6marks)**

```
TPE_algo <- function(train_data, valid_data, max_iter=100) {
  #Initialise with random lambda and calculate the loss
  lambda_star = sample_lambdas(1)
  loss_z_star = my_loss(lambda_star, train_data, valid_data)

  # Define evaluation points for KDE (same for both g and b)
  x_values <- seq(0, 1, length.out = 100)

  for (iter in max_iter){
    #Generate 100 ~ U[0,1]
    sample_lambdas = sample_lambdas(100)

    #Compute loss for each lambda
    losses <- sapply(sample_lambdas, function(lambda) my_loss(lambda, train_data, valid_data))

    #Store good losses and bad losses
    good_losses <- which(losses <= loss_z_star)
    bad_losses <- which(losses > loss_z_star)

    #Skip iteration if fewer than 2 values exist in either froups
    if (length(good_losses) < 2 || length(bad_losses) < 2){
      next #Skip iteration
    }
```

```r
    #KDE estimation for g(lambda) and b(lambda)

    optimise_function <- function(x_values) {
      g_lambda <- my_kde(sample_lambdas[good_losses], x_values)
      b_lambda <- my_kde(sample_lambdas[bad_losses], x_values)
      #Ratio to minimised later
      ratio = b_lambda/g_lambda
      return(ratio)
    }

    #Calculate new lambda star value which minimise the ratio
    lambda_star <- optimise(optimise_function, interval = c(min(x_values), max(x_values)))$minimum

    #Calculate new loss_z_star value
    loss_z_star <- my_loss(lambda_star, train_data, valid_data)

  }
  return(lambda_star)
}
```

(vii) **(2marks)**

```r
set.seed(42)
lambda_star <- TPE_algo(train_Data, valid_Data)
#Calculate test mean square error
test_mse <- my_loss(lambda_star, train_Data, test_Data)

cat("Final value of lambda_star:", lambda_star, "\n")
```

```
## Final value of lambda_star: 0.914806
```

```r
cat("The test MSE is:", test_mse, "\n")
```

```
## The test MSE is: 1.028833
```