Stat 432 Homework 6

Assigned: Sep 29, 2024; Due: 11:59 PM CT, Oct 10, 2024

Contents

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope. No email or hard copy will be accepted. For **late submission policy and grading rubrics**, please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.
- **When using AI tools**, you are encouraged to document your comment on your experience with AI tools especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is $\geq$ 4.0.0. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

Question 1: Multivariate Kernel Regression Simulation (45 pts)

Similar to the previous homework, we will use simulated datasets to evaluate a kernel regression model. You should write your own code to complete this question. We use two-dimensional data generator:

$$Y = \exp(\beta^T x) + \epsilon$$

where $\beta = c(1, 1)$, X is generated uniformly from $[0, 1]^2$, and ϵ follows i.i.d. standard Gaussian. Use the following code to generate a set of training and testing data:

```
  set.seed(2)
  trainn <- 200
  testn <- 1
  p = 2
  beta <- c(1.5, 1.5)

  # generate data

  Xtrain <- matrix(runif(trainn * p), ncol = p)
  Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)
  Xtest <- matrix(runif(testn * p), ncol = p)

  # the first testing observation
  Xtest
```

```
##           [,1]      [,2]
## [1,] 0.4152441 0.5314388
```

```
  # the true expectation of the first testing observation
  exp(Xtest %*% beta)
```

```
##          [,1]
## [1,] 4.137221
```

a. [10 pts] For this question, you need to **write your own code** for implementing a two-dimensional Nadaraya-Watson kernel regression estimator, and predict **just the first testing observation**. For this task, we will use independent Gaussian kernel function introduced during the lecture. Use the same bandwidth h for both dimensions. As a starting point, use $h = 0.07$. What is your predicted value?

```
mu0 <- exp(Xtest %*% beta)
h <- 0.07

# Compute squared distances between Xtest and Xtrain
diffs <- sweep(Xtrain, 2, Xtest[1, ])
sq_dists <- rowSums(diffs^2)

# Compute kernel weights using the Gaussian kernel
wts <- exp(-sq_dists / (2 * h^2))

# Compute the Nadaraya-Watson estimator
Yhat <- sum(wts * Ytrain) / sum(wts)
Yhat
```

```
## [1] 4.198552
```

b. [20 pts] Based on our previous understanding the bias-variance trade-off of KNN, do the same simulation analysis for the kernel regression model. Again, you only need to consider the predictor of this one testing point. Your simulation needs to be able to calculate the following quantities:

- Bias^2

- Variance
- Mean squared error (MSE) of prediction

Use at least 5000 simulation runs. Based on your simulation, answer the following questions:

- Does the MSE matches our theoretical understanding of the bias-variance trade-off?
- Comparing the bias and variance you have, should we increase or decrease the bandwidth h to reduce the MSE?

```r
# Set parameters
set.seed(2)
trainn <- 200
testn <- 1
p <- 2
beta <- c(1.5, 1.5)
h <- 0.07
simulations <- 5000

# Testing point
Xtest <- matrix(c(0.4152441, 0.5314388), nrow = 1)  # Provided test point
true_value <- exp(Xtest %*% beta)  # True expected value

# Gaussian Kernel function
gaussian_kernel <- function(x, x_i, h) {
  exp(-sum((x - x_i)^2) / (2 * h^2))
}

# Nadaraya-Watson Kernel Regression Estimator
nw_kernel_regression <- function(Xtrain, Ytrain, Xtest, h) {
  num <- 0
  denom <- 0
  for (i in 1:nrow(Xtrain)) {
    kernel_value <- gaussian_kernel(Xtest, Xtrain[i, ], h)
    num <- num + kernel_value * Ytrain[i]
    denom <- denom + kernel_value
  }
  return(num / denom)
}

# Simulate 5000 times
predictions <- numeric(simulations)
for (sim in 1:simulations) {
  # Generate training data for each simulation
  Xtrain <- matrix(runif(trainn * p), ncol = p)
  Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)

  # Predict using Nadaraya-Watson kernel regression
  predictions[sim] <- nw_kernel_regression(Xtrain, Ytrain, Xtest, h)
}

# Calculate Bias^2
bias_squared <- (mean(predictions) - true_value)^2
```

```r
# Calculate Variance
variance <- var(predictions)

# Calculate MSE
mse <- mean((predictions - true_value)^2)
```

```
## Warning in predictions - true_value: Recycling array of length 1 in vector-array arithmetic is depre
##   Use c() or as.vector() instead.
```

```r
# Output results
cat("Bias^2:", bias_squared, "\n")
```

```
## Bias^2: 0.002284004
```

```r
cat("Variance:", variance, "\n")
```

```
## Variance: 0.09560785
```

```r
cat("Mean Squared Error (MSE):", mse, "\n")
```

```
## Mean Squared Error (MSE): 0.09787273
```

Yes, the MSE is approximately equal to the sum of the bias squared and the variance. Since the variance is significantly larger than the bias squared, increasing the bandwidth h would reduce the variance more than it would increase the bias squared, potentially reducing the MSE. Therefore, we should increase the bandwidth to reduce the MSE.

c. [15 pts] In practice, we will have to use cross-validation to select the optimal bandwidth. However, if you have the power of simulating as many datasets as you can, and you also know the true model, how would you find the optimal bandwidth for the bias-variance trade-off for this particular model and sample size? Provide enough evidence to claim that your selected bandwidth is (almost) optimal.

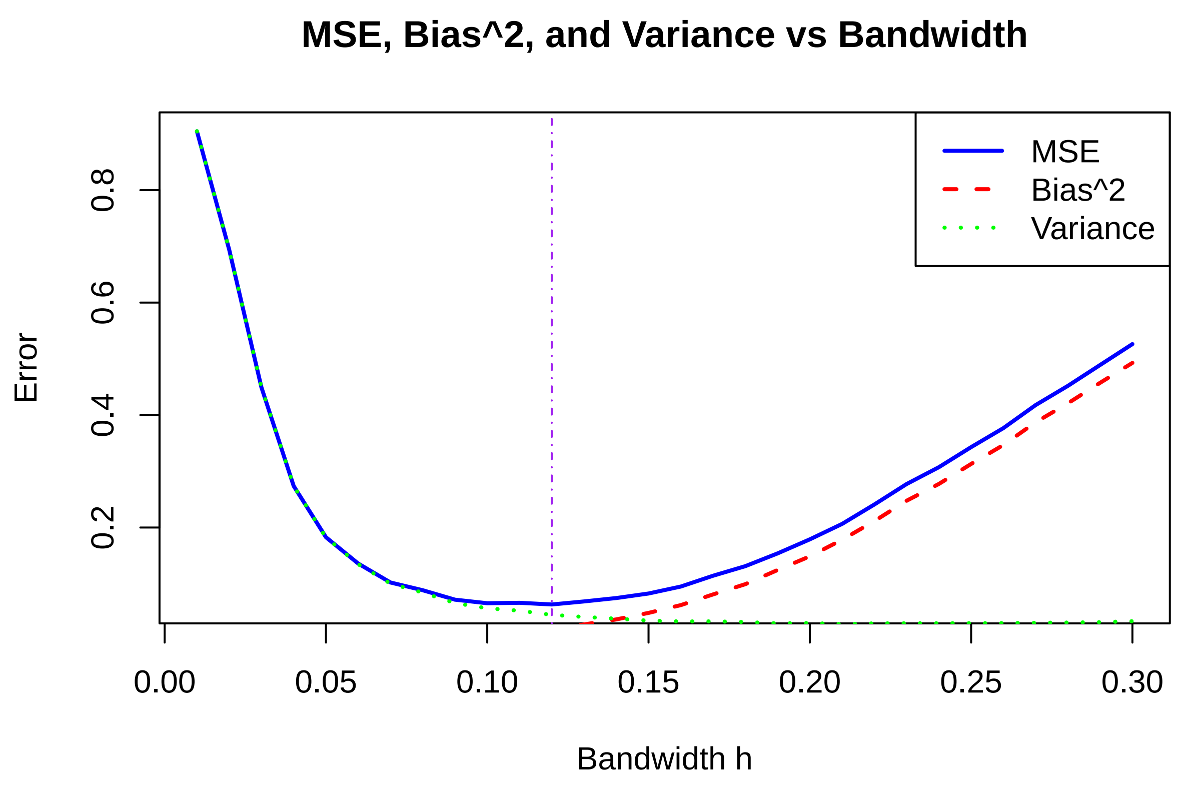
```r
# Set the seed for reproducibility
set.seed(2)

# Parameters
n_simulations <- 5000
trainn <- 200
p <- 2
beta <- c(1.5, 1.5)

# Fixed test point
Xtest <- matrix(c(0.4152441, 0.5314388), ncol = p)

# True expectation at the test point
true_expectation <- as.numeric(exp(Xtest %*% beta))

# Define a sequence of bandwidths to test
h_values <- seq(0.01, 0.3, by = 0.01)
```

```r
# Initialize vectors to store results
MSE_values <- numeric(length(h_values))
Bias_squared_values <- numeric(length(h_values))
Variance_values <- numeric(length(h_values))

# Loop over each bandwidth
for (j in seq_along(h_values)) {
  h <- h_values[j]

  # Initialize a vector to store predictions
  predictions <- numeric(n_simulations)

  # Simulation loop
  for (i in 1:n_simulations) {
    # Generate training data
    Xtrain <- matrix(runif(trainn * p), ncol = p)
    Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)

    # Compute scaled differences
    u1 <- (Xtest[1,1] - Xtrain[,1]) / h
    u2 <- (Xtest[1,2] - Xtrain[,2]) / h

    # Compute weights using Gaussian kernel
    w <- exp(-0.5 * (u1^2 + u2^2))

    # Avoid division by zero
    if (sum(w) == 0) {
      predictions[i] <- NA
    } else {
      # Compute the estimator
      numerator <- sum(w * Ytrain)
      denominator <- sum(w)
      predictions[i] <- numerator / denominator
    }
  }

  # Remove NA values (if any)
  predictions <- predictions[!is.na(predictions)]

  # Calculate bias^2, variance, and MSE
  mean_prediction <- mean(predictions)
  bias_squared <- (mean_prediction - true_expectation)^2
  variance <- var(predictions)
  MSE <- mean((predictions - true_expectation)^2)

  # Store the results
  Bias_squared_values[j] <- bias_squared
  Variance_values[j] <- variance
  MSE_values[j] <- MSE
}

# Find the optimal bandwidth
optimal_index <- which.min(MSE_values)
```

```
optimal_h <- h_values[optimal_index]
cat("Optimal bandwidth h*:", optimal_h, "\n")
```

```
## Optimal bandwidth h*: 0.12
```

```
# Plot MSE vs h
plot(h_values, MSE_values, type = "l", col = "blue", lwd = 2,
     xlab = "Bandwidth h", ylab = "Error",
     main = "MSE, Bias^2, and Variance vs Bandwidth")
lines(h_values, Bias_squared_values, col = "red", lwd = 2, lty = 2)
lines(h_values, Variance_values, col = "green", lwd = 2, lty = 3)
legend("topright", legend = c("MSE", "Bias^2", "Variance"),
       col = c("blue", "red", "green"), lty = c(1,2,3), lwd = 2)
abline(v = optimal_h, col = "purple", lty = 4)
```



We found that the optimal bandwidth is h: 0.12 The plot shows that as h increases, the variance decreases while the bias squared increases. The MSE curve reaches its minimum at h = 0.12, indicating the optimal balance. Beyond h=0.1131579., the increase in bias squared outweighs the reduction in variance, causing the MSE to rise.

Question 2: Local Polynomial Regression (55 pts)

We introduced the local polynomial regression in the lecture, with the objective function for predicting a target point x_0 defined as

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_{x_0})^{\mathrm{T}}\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_{x_0}),$$

where W is a diagonal weight matrix, with the ith diagonal element defined as $K_h(x_0, x_i)$, the kernel distance between x_i and x_0. In this question, we will write our own code to implement this model. We will use the same simulated data provided at the beginning of Question 1.

```
  set.seed(2)
  trainn <- 200
  testn <- 1
  p = 2
  beta <- c(1.5, 1.5)

  # generate data

  Xtrain <- matrix(runif(trainn * p), ncol = p)
  Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)
  Xtest <- matrix(runif(testn * p), ncol = p)
```

a. [10 pts] Using the same kernel function as Question 1, calculate the kernel weights of x_0 against all observed training data points. Report the 25th, 50th and 75th percentiles of the weights so we can check your answer.

```
# Bandwidth
h <- 0.07

# Compute differences between Xtest and Xtrain
diffs <- sweep(Xtrain, 2, Xtest[1, ])
sq_dists <- rowSums(diffs^2)

# Compute kernel weights
wts <- exp(-sq_dists / (2 * h^2))

weight_percentiles <- quantile(wts, probs = c(0.25, 0.5, 0.75))
weight_percentiles
```

```
##          25%          50%          75%
## 5.774659e-13 4.999524e-08 3.882088e-05
```

b. [15 pts] Based on the objective function, derive the normal equation for estimating the local polynomial regression in matrix form. And then define the estimated $\boldsymbol{\beta}_{x_0}$. Write your answer in latex.

Derivation

To find β_{x_0} that minimizes the objective function, we take the derivative with respect to β_{x_0} and set it to zero:

$$\frac{\partial}{\partial \beta_{x_0}} \left((\mathbf{y} - \mathbf{X}\beta_{x_0})^\top \mathbf{W} (\mathbf{y} - \mathbf{X}\beta_{x_0}) \right) = 0.$$

Computing the derivative:

$$-2\mathbf{X}^\top \mathbf{W}(\mathbf{y} - \mathbf{X}\beta_{x_0}) = 0.$$

Simplifying:

$$\mathbf{X}^\top \mathbf{W}\mathbf{X}\beta_{x_0} = \mathbf{X}^\top \mathbf{W}\mathbf{y}.$$

Normal Equation

Therefore, the normal equation for estimating β_{x_0} is:

$$\beta_{x_0} = \left(\mathbf{X}^\top \mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}\mathbf{y}.$$

c. [10 pts] Based on the observed data provided in Question 1, calculate the estimated $\boldsymbol{\beta}_{x_0}$ for the testing point `Xtest` using the formula you derived. Report the estimated $\boldsymbol{\beta}_{x_0}$. Calculate the prediction on the testing point and compare it with the true expectation.

```r
# Construct weight matrix W
W <- diag(wts)

# Add intercept to Xtrain
Xtrain_intercept <- cbind(1, Xtrain)

# Compute (X^T W X)
XTWX <- t(Xtrain_intercept) %*% W %*% Xtrain_intercept

# Compute (X^T W y)
XTWy <- t(Xtrain_intercept) %*% W %*% Ytrain

# Solve for beta_x0
beta_x0 <- solve(XTWX, XTWy)
beta_x0
```

```
##           [,1]
## [1,] -1.749993
## [2,]  5.690594
## [3,]  6.870116
```

```r
# Add intercept to Xtest
Xtest_intercept <- cbind(1, Xtest)

# Prediction at Xtest
Yhat <- Xtest_intercept %*% beta_x0
Yhat
```

```
##          [,1]
## [1,] 4.264039
```

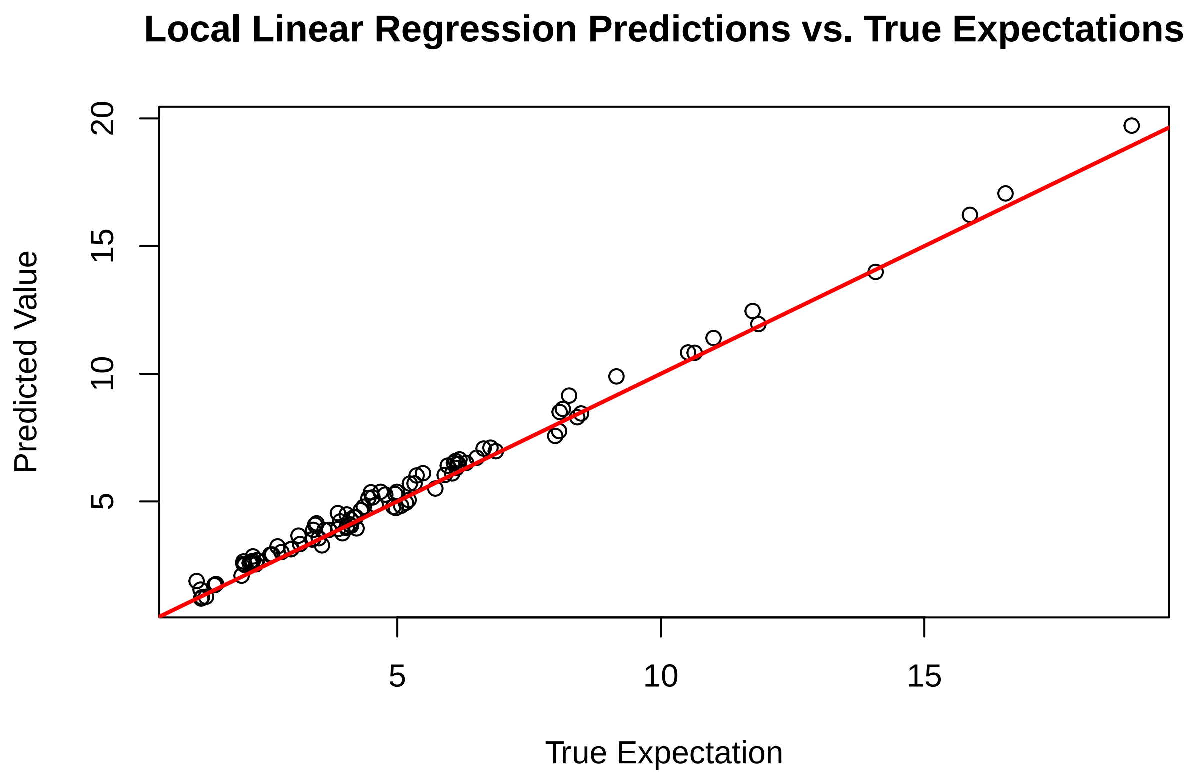
```r
mu0 <- exp(Xtest %*% beta)
mu0
```

```
##          [,1]
## [1,] 4.137221
```

d. [20 pts] Now, let's use this model to predict the following 100 testing points. After you fit the model, provide a scatter plot of the true expectation versus the predicted values on these testing points. Does this seem to be a good fit? As a comparison, fit a global linear regression model to the training data and predict the testing points. Does your local linear model outperforms the global linear mode? Note: this is not a simulation study. You should use the same training data provided previously.

```
  set.seed(432)
  testn <- 100
  Xtest <- matrix(runif(testn * p), ncol = p)
```

```
# Initialize vectors to store predictions
Yhat_local <- numeric(testn)
mu_true <- numeric(testn)

# Loop over each test point
for (i in 1:testn) {
  # Current test point
  x0 <- Xtest[i, ]

  # Compute differences and weights
  diffs <- sweep(Xtrain, 2, x0)
  sq_dists <- rowSums(diffs^2)
  wts <- exp(-sq_dists / (2 * h^2))

  # Weight matrix
  W <- diag(wts)

  # Local design matrix with intercept
  Xtrain_intercept <- cbind(1, Xtrain)

  # Compute beta_x0
  XTWX <- t(Xtrain_intercept) %*% W %*% Xtrain_intercept
  XTWy <- t(Xtrain_intercept) %*% W %*% Ytrain
  beta_x0 <- solve(XTWX, XTWy)

  # Prediction at x0
  x0_intercept <- c(1, x0)
  Yhat_local[i] <- x0_intercept %*% beta_x0

  # True expectation
  mu_true[i] <- exp(x0 %*% beta)
}

# Plotting
plot(mu_true, Yhat_local, xlab = "True Expectation", ylab = "Predicted Value",
     main = "Local Linear Regression Predictions vs. True Expectations")
abline(0, 1, col = "red", lwd = 2)
```



```r
train_data <- data.frame(
  Ytrain = Ytrain,
  X1 = Xtrain[, 1],
  X2 = Xtrain[, 2]
)

global_model <- lm(Ytrain ~ X1 + X2, data = train_data)

test_data <- data.frame(
  X1 = Xtest[, 1],
  X2 = Xtest[, 2]
)

Yhat_global <- predict(global_model, newdata = test_data)

# Calculating MSE for both models
# MSE for local linear regression
mse_local <- mean((Yhat_local - mu_true)^2)
cat("Local Linear Regression MSE:", mse_local, "\n")
```

```
## Local Linear Regression MSE: 0.1534425
```

```r
# MSE for global linear regression
mse_global <- mean((Yhat_global - mu_true)^2)
cat("Global Linear Regression MSE:", mse_global, "\n")
```

```
## Global Linear Regression MSE: 1.751315
```

The plot shows that this is a good fit. The local linear regression model provides a much better fit compared to the global linear regression model for this dataset.