Stat 432 Homework 6 Jerry Liang

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

Contents

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?

```
gaussian_kernel <- function(x, xi, h) {
  return (exp(-sum((x - xi)^2) / (2 * h^2)) / ((2 * pi * h^2)^(p / 2)))
}

# Nadaraya-Watson kernel regression estimator
nadaraya_watson <- 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)
}

# Predict the first testing observation using the Nadaraya-Watson estimator
predicted_value <- nadaraya_watson(Xtrain, Ytrain, Xtest, h=0.07)
predicted_value
```

```
## [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?

```
set.seed(2)
trainn <- 200
testn <- 1
p <- 2
beta <- c(1.5, 1.5)
h <- 0.07
num_simulations <- 5000

predictions <- numeric(num_simulations)
```

```r
# The true expectation of the first testing observation
true_value <- as.vector(exp(Xtest %*% beta))

# Simulation loop
for (sim in 1:num_simulations) {
  # Generate training data
  Xtrain <- matrix(runif(trainn * p), ncol = p)
  Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)
  predictions[sim] <- nadaraya_watson(Xtrain, Ytrain, Xtest, h)
}

# Calculate Bias^2, Variance, and MSE
bias_squared <- (mean(predictions) - true_value)^2
variance <- var(predictions)
mse <- mean((predictions - true_value)^2)

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

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

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

```
## Variance:  0.09560785
```

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

```
## MSE:  0.09787273
```

Yes, the result of MSE and variance bias trade off generally speaking, match the theoritical result of MSE= Bias^2 + Variance + noise according to its value. Since varince is much larger than bias^2, in this case, we should increase the bandwidth h. Increasing h will smooth the model, reduce its sensitivity to small variations in the data, and lower the variance.

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.

We'll iterate over a range of bandwidth values, perform simulations for each value, and then calculate the average MSE to find the optimal bandwidth.

```r
set.seed(2)
trainn <- 200
p <- 2
beta <- c(1.5, 1.5)
num_simulations <- 5000
bandwidth_values <- seq(0.01, 0.2, by = 0.01)
mse_results <- numeric(length(bandwidth_values))
```

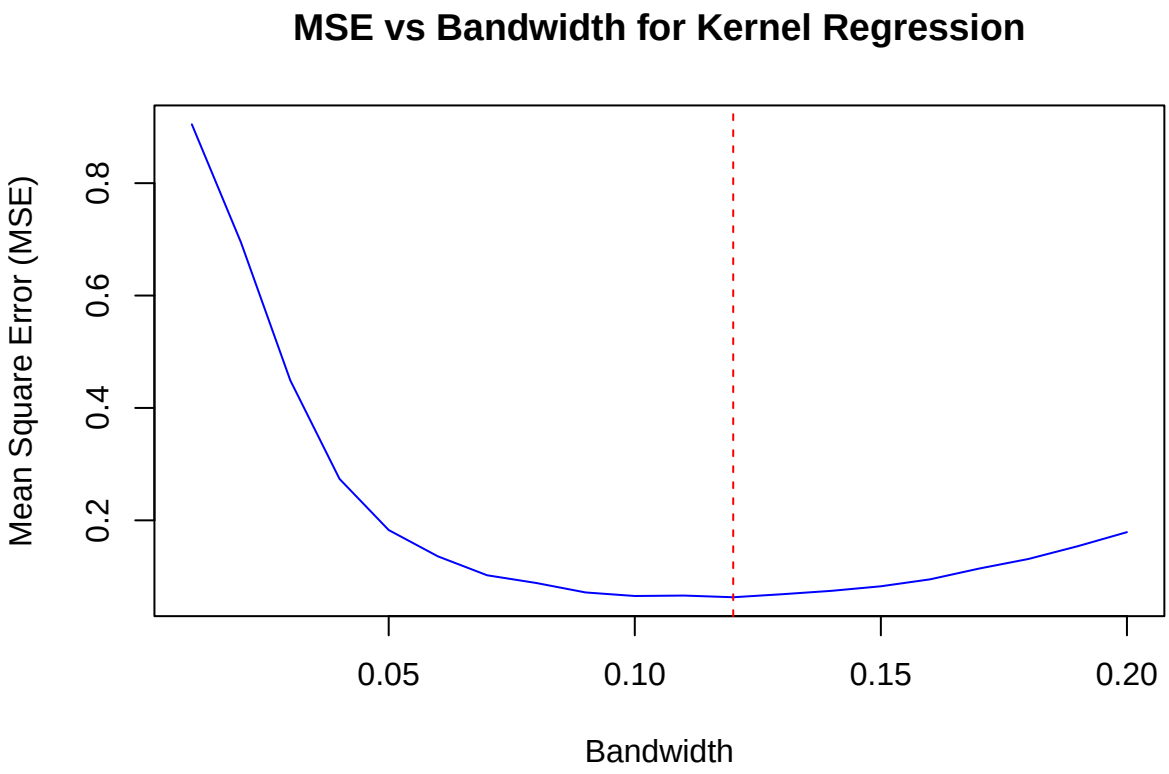
```r
# The true expectation of the first testing observation
Xtest <- matrix(c(0.4152441, 0.5314388), nrow = 1)
true_value <- as.vector(exp(Xtest %*% beta))

# Function to compute the MSE for a given bandwidth
compute_mse_for_bandwidth <- function(h) {
  predictions <- numeric(num_simulations)

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

    # Predict using the Nadaraya-Watson estimator
    predictions[sim] <- nadaraya_watson(Xtrain, Ytrain, Xtest, h)
  }

  # Calculate MSE for this bandwidth
  mse <- mean((predictions - true_value)^2)
  return(mse)
}

# Loop through each bandwidth and compute the MSE
for (i in seq_along(bandwidth_values)) {
  mse_results[i] <- compute_mse_for_bandwidth(bandwidth_values[i])
}

# Find the optimal bandwidth
optimal_bandwidth <- bandwidth_values[which.min(mse_results)]
cat("Optimal Bandwidth:", optimal_bandwidth, "\n")
```

```
## Optimal Bandwidth: 0.12
```

```r
cat("Minimum MSE:", min(mse_results), "\n")
```

```
## Minimum MSE: 0.06318511
```

```r
# Plot MSE against Bandwidth
plot(bandwidth_values, mse_results, type = "l", col = "blue",
     xlab = "Bandwidth", ylab = "Mean Square Error (MSE)",
     main = "MSE vs Bandwidth for Kernel Regression")
abline(v = optimal_bandwidth, col = "red", lty = 2)
```



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.

```
weights <- sapply(1:trainn, function(i) gaussian_kernel(Xtest, Xtrain[i, ], h))

percentiles <- quantile(weights, probs = c(0.25, 0.5, 0.75))
percentiles
```

```
##          25%          50%          75%
## 1.875644e-11 1.623875e-06 1.260925e-03
```

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.

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

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_{x_0})^{\mathrm{T}}\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}_{x_0}) = \mathbf{y}^{\mathrm{T}}\mathbf{W}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{W}\mathbf{X}\boldsymbol{\beta}_{x_0} + \boldsymbol{\beta}_{x_0}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X}\boldsymbol{\beta}_{x_0}.$$

$$\frac{\partial}{\partial \boldsymbol{\beta}_{x_0}}\left(\mathbf{y}^{\mathrm{T}}\mathbf{W}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{W}\mathbf{X}\boldsymbol{\beta}_{x_0} + \boldsymbol{\beta}_{x_0}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X}\boldsymbol{\beta}_{x_0}\right) = 0.$$

$$-2\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{y} + 2\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X}\boldsymbol{\beta}_{x_0} = 0.$$

$$\boldsymbol{\beta}_{x_0} = (\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{y}$$

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.

```
weights <- sapply(1:trainn, function(i) gaussian_kernel(Xtest, Xtrain[i, ], h))
W <- diag(weights)  # Diagonal weight matrix

X_design <- cbind(1, Xtrain)

beta_x0_estimated <- solve(t(X_design) %*% W %*% X_design) %*% t(X_design) %*% W %*% Ytrain

cat("Estimated beta_x0: ", beta_x0_estimated, "\n")
```

```
## Estimated beta_x0:  -1.749993 5.690594 6.870116
```

```
Xtest_design <- c(1, Xtest)
prediction <- sum(Xtest_design * beta_x0_estimated)

# True expectation of the testing point
true_expectation <- exp(Xtest %*% beta)

# Display the results
cat("Prediction on the testing point: ", prediction, "\n")
```

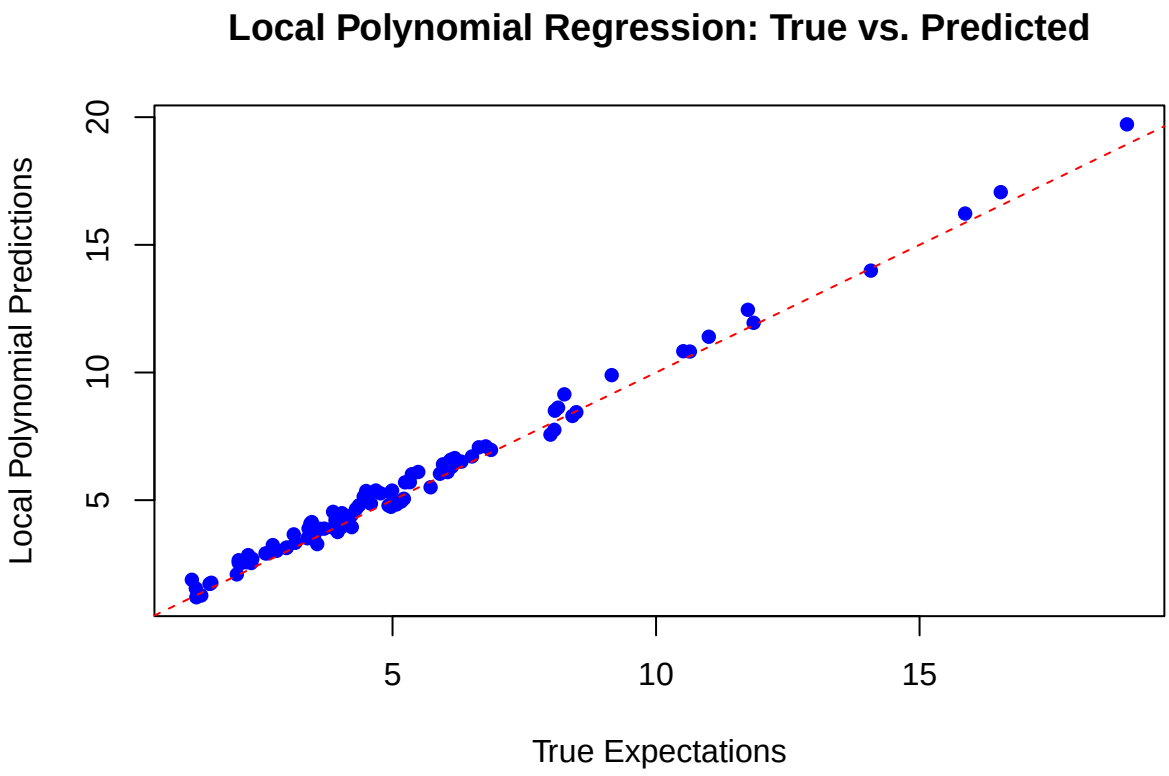
```
## Prediction on the testing point:  4.264039
```

```r
cat("True expectation of the testing point: ", true_expectation, "\n")
```

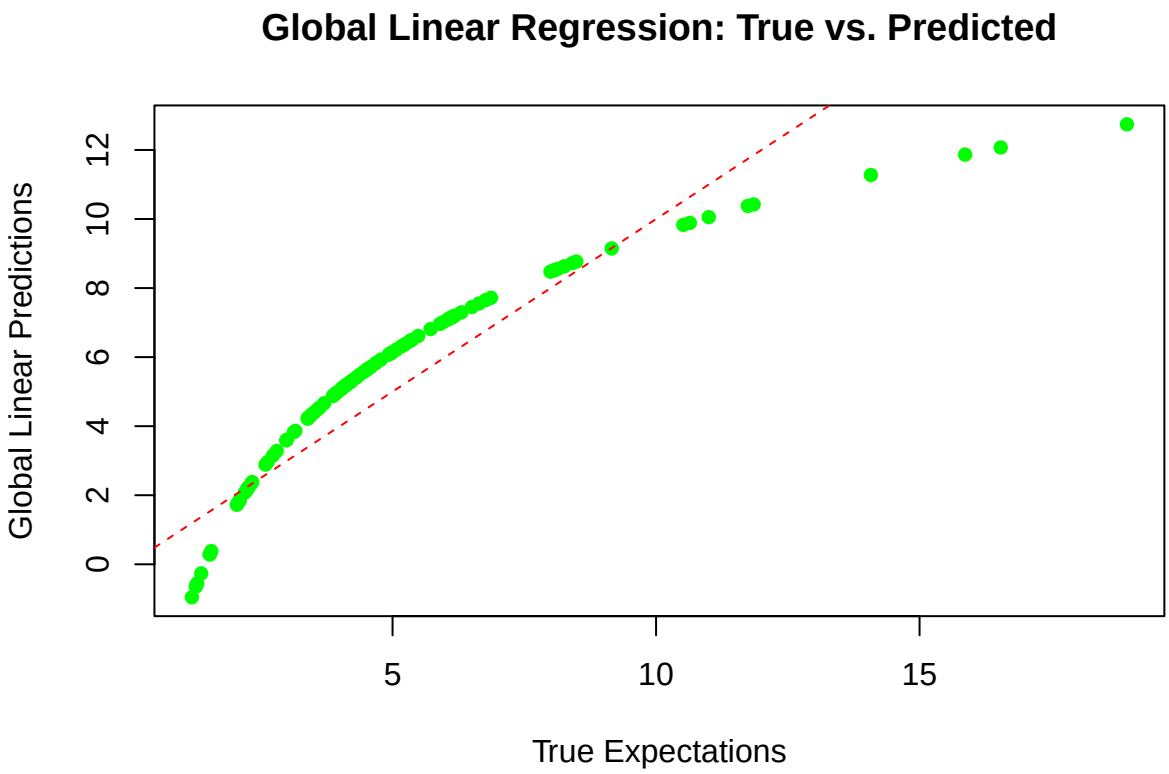
```
## True expectation of the testing point:  4.137221
```

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. This seems to be a good fit, and the local linear model outperforms the global linear model.

```r
  set.seed(432)
  testn <- 100
  Xtest_all <- matrix(runif(testn * p), ncol = p)
true_expectations <- exp(Xtest_all %*% beta)
local_polynomial_prediction <- function(Xtrain, Ytrain, Xtest_point, h) {
  weights <- sapply(1:trainn, function(i) gaussian_kernel(Xtest_point, Xtrain[i, ], h))
  W <- diag(weights)
  X_design <- cbind(1, Xtrain)
  beta_x0_estimated <- solve(t(X_design) %*% W %*% X_design) %*% t(X_design) %*% W %*% Ytrain
  Xtest_design <- c(1, Xtest_point)
  prediction <- sum(Xtest_design * beta_x0_estimated)
  return(prediction)
}
local_predictions <- sapply(1:testn, function(i) local_polynomial_prediction(Xtrain, Ytrain, Xtest_all[

global_model <- lm(Ytrain ~ Xtrain)
Xtest_all_df <- data.frame(Xtrain = I(Xtest_all))
# Predict the target values for the 100 testing points using the global model
global_predictions <- predict(global_model, newdata = Xtest_all_df)

plot(true_expectations, local_predictions,
     xlab = "True Expectations", ylab = "Local Polynomial Predictions",
     main = "Local Polynomial Regression: True vs. Predicted",
     col = "blue", pch = 16)
abline(0, 1, col = "red", lty = 2)  # Line y = x for reference
```




```r
# Plot true expectations vs. global linear predictions
plot(true_expectations, global_predictions,
     xlab = "True Expectations", ylab = "Global Linear Predictions",
     main = "Global Linear Regression: True vs. Predicted",
     col = "green", pch = 16)
abline(0, 1, col = "red", lty = 2)
```


Global Linear Regression: True vs. Predicted
Global Linear Predictions
0
2
4
6
8
10
12
5
10
15
True Expectations