Stat 432 Homework 6

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

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 (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/courses/570816). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), 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?

Solution:

```
  # Calculate the Euclidean distance between all training data and the first testing data
  h = 0.07
  w <- exp(-rowSums(sweep(Xtrain, 2, Xtest, "-")^2)/h^2/2) # skipping the constant term since it
doesnt matter
  yhat <- sum(w * Ytrain) / sum(w)
  yhat
```

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

```
  # compare with the true mean
  yhat - exp(Xtest %*% beta)
```

```
##            [,1]
## [1,] 0.06133096
```

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?

Solution:

```
# Calculate the bias, variance, and MSE
nsim <- 5000
yhat <- numeric(nsim)
errors <- numeric(nsim)

for (i in 1:nsim) {
  Xtrain <- matrix(runif(trainn * p), ncol = p)
  Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)
  w <- exp(-rowSums(sweep(Xtrain, 2, Xtest, "-")^2)/h^2/2)
  yhat[i] <- sum(w * Ytrain) / sum(w)

  # ytest
  ytest = exp(Xtest %*% beta) + rnorm(1)
  errors[i] <- yhat[i] - ytest
}

bias <- mean(yhat) - exp(Xtest %*% beta)
variance <- var(yhat)
mse <- mean(errors^2)

# report results and validate our theory
bias^2
```

```
##             [,1]
## [1,] 0.001825033
```

```
variance
```

```
## [1] 0.09545267
```

```
bias^2 + variance + 1
```

```
##          [,1]
## [1,] 1.097278
```

```
mse
```

```
## [1] 1.076495
```

Based on our theory, MSE should be the sum of bias^2, variance, and irreducible error. In this case, the MSE is 1.0972777, which is very close to the simulated MSE (1.0764953). This confirms our theory. In terms of bias and variance, the variance is much larger. Hence we should increase the bandwidth h 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.

Solution:

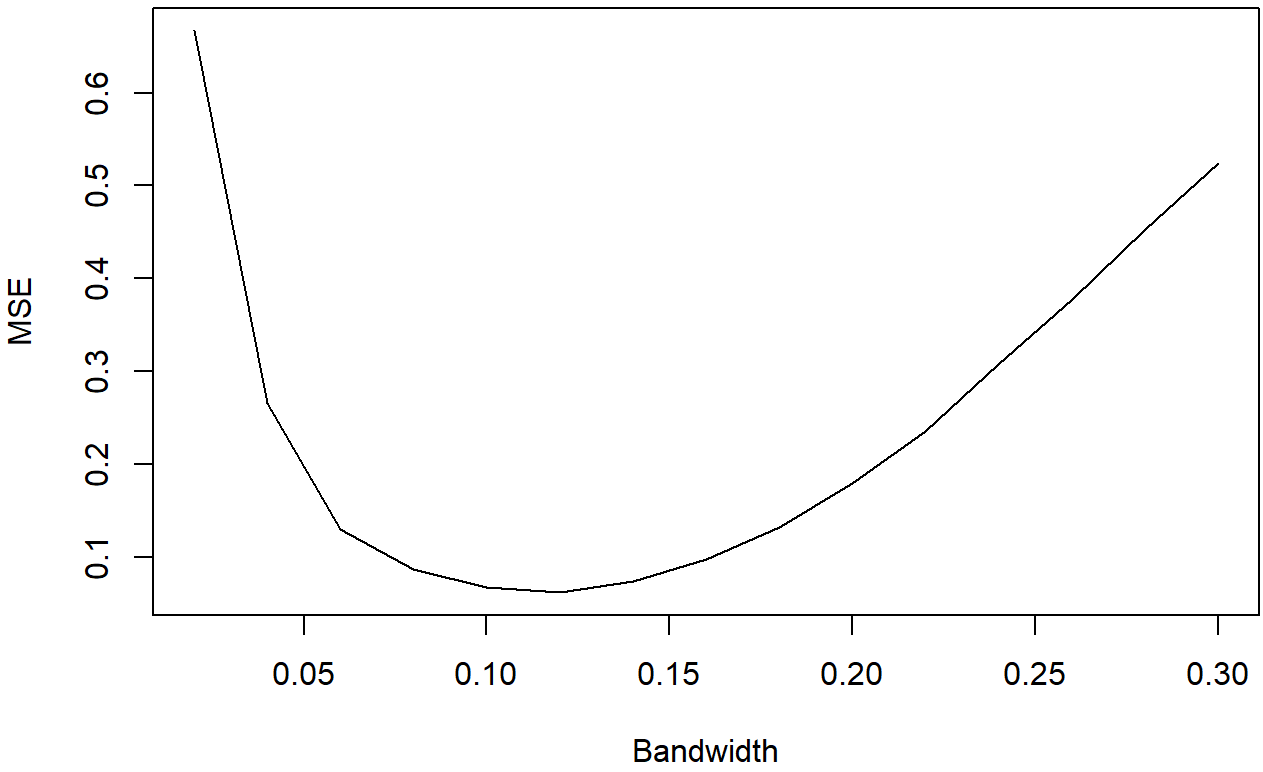
```
# define the bandwidth to consider
hseq <- seq(0.02, 0.3, by = 0.02)
biasall <- numeric(length(hseq))
varianceall <- numeric(length(hseq))

nsim <- 5000
# Calculate the prediction error
for (j in 1:length(hseq)) {
  h <- hseq[j]

  for (i in 1:nsim) {
    Xtrain <- matrix(runif(trainn * p), ncol = p)
    Ytrain <- exp(Xtrain %*% beta) + rnorm(trainn)
    w <- exp(-rowSums(sweep(Xtrain, 2, Xtest, "-")^2)/h^2/2)
    yhat[i] <- sum(w * Ytrain) / sum(w)
  }

  biasall[j] <- mean(yhat) - exp(Xtest %*% beta)
  varianceall[j] <- var(yhat)
}

# plot the MSE
plot(hseq, biasall^2 + varianceall, type = "l", xlab = "Bandwidth", ylab = "MSE")
```




```
# find the optimal bandwidth
hseq[which.min(biasall^2 + varianceall)]
```

```
## [1] 0.12
```

Our simulation seems to stable enough and with a fine grid of bandwidths, we can see that the MSE is minimized at 0.12. This bandwidth is (almost) optimal for this particular model and sample size.

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)

# 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] 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.

Solution:

```
# Calculate the weight matrix
h = 0.07
W <- diag(exp(-rowSums(sweep(Xtrain, 2, Xtest, "-")^2)/h^2/2) / (2*pi*h) )
quantile(diag(W), c(0.25, 0.5, 0.75))
```

```
##          25%          50%          75%
## 1.312951e-12 1.136713e-07 8.826478e-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.

Solution:

The normal equation for estimating the local polynomial regression in matrix form is:

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

Hence, the estimated $\boldsymbol{\beta}_{x_0}$ is:

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

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.

Solution:

```
  # Calculate the estimated beta
  beta_hat <- solve(t(Xtrain) %*% W %*% Xtrain) %*% t(Xtrain) %*% W %*% Ytrain
  beta_hat
```

```
##          [,1]
## [1,] 3.978612
## [2,] 4.907357
```

```
  # Predict the testing point
  yhat <- Xtest %*% beta_hat

  # the truth
  exp(Xtest %*% beta)
```

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

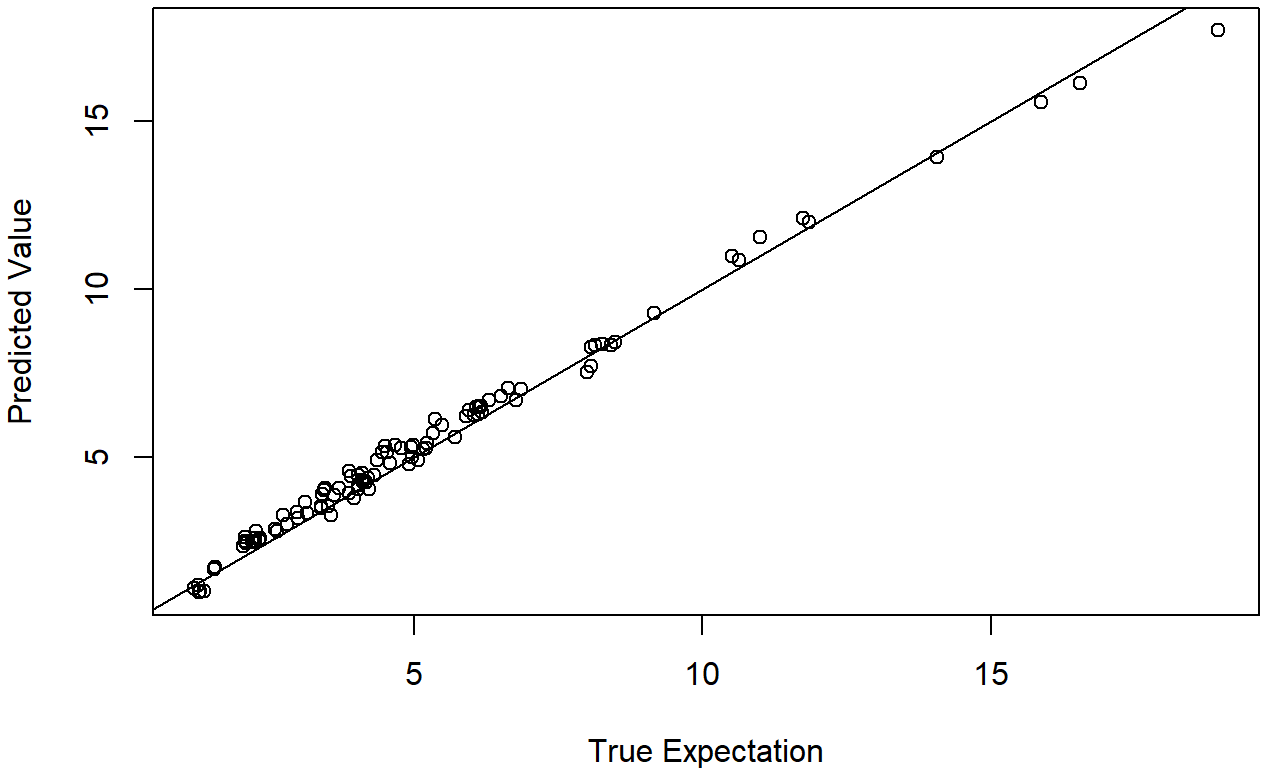
d. [20 pts] Now, let's use this model to predict the following 100 testing points. After you fit the model, provide a scatter plot of the true expectation versus the predicted values on these testing points. Does this seem to be a good fit? As a comparison, fit a global linear regression model to the 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)
```

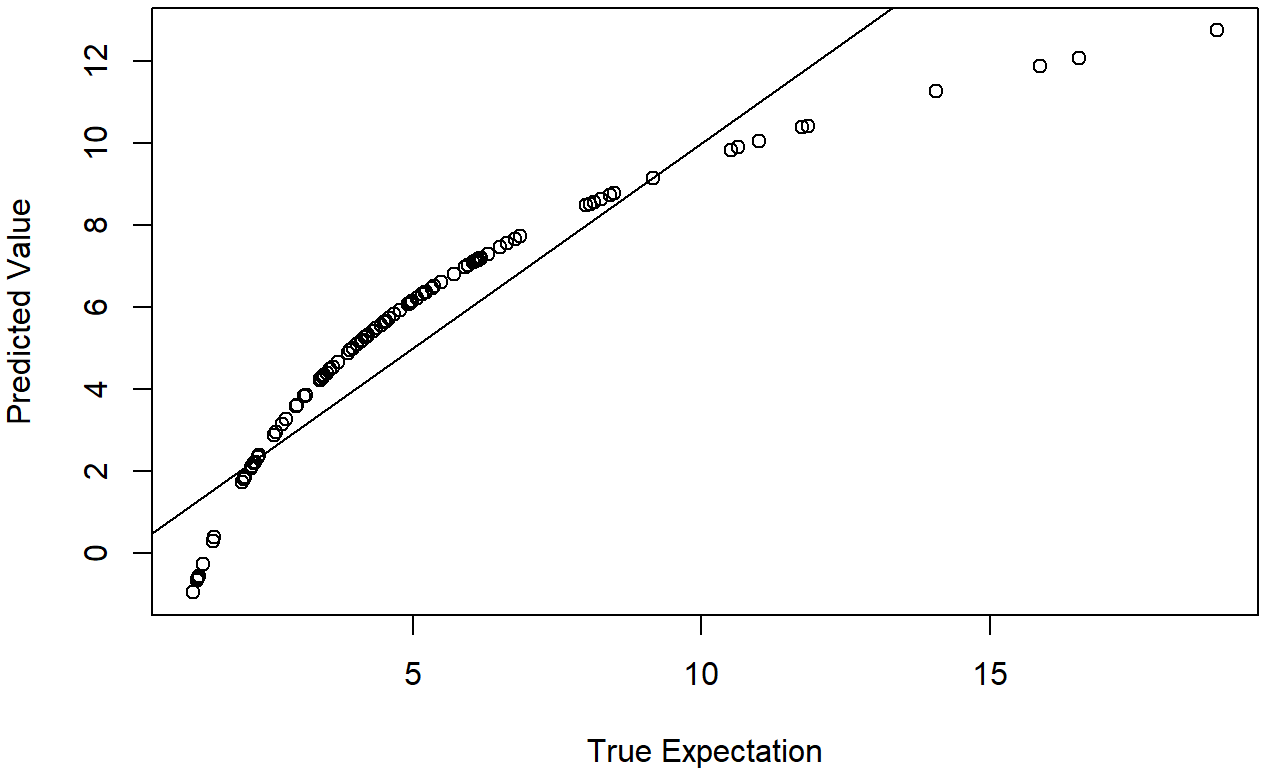
Solution:

```
  # Predict the testing points
  yhat <- numeric(testn)
  for (i in 1:testn) {
    W <- diag(exp(-rowSums(sweep(Xtrain, 2, Xtest[i,], "-")^2)/h^2/2) / (2*pi*h) )
    beta_hat <- solve(t(Xtrain) %*% W %*% Xtrain) %*% t(Xtrain) %*% W %*% Ytrain
    yhat[i] <- Xtest[i,] %*% beta_hat
  }

  ymean <- exp(Xtest %*% beta)
  plot(ymean, yhat, xlab = "True Expectation", ylab = "Predicted Value")
  abline(coef = c(0,1))
```




```
# Fit a global linear regression model
lmfit <- lm(y ~ ., data = data.frame(Xtrain, "y" = Ytrain))
yhat_lm <- predict(lmfit, data.frame(Xtest))
plot(ymean, yhat_lm, xlab = "True Expectation", ylab = "Predicted Value")
abline(coef = c(0,1))
```


Predicted Value
0
2
4
6
8
10
12
5
10
15
True Expectation