STAT 542: Homework 6

Spring 2022, by Kamila Makhambetova (kamilam3)

Due: Monday, March 10, 11:59 PM CT

Instruction

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code 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 compass2g. No email or hardcopy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat542/homework.html), please refer to the course website.

- What is expected for the submission to **Gradescope**
    - You are required to submit one rendered **PDF** file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file generated by a `.Rmd` file. `.html` format cannot be accepted.
    - Please follow the instructions on Gradescope to select corresponding PDF pages for each question.
- Please note that your homework file is a **PDF** report instead of a messy collection of R codes. This report should **include**:
    - Your Name and NetID. (Replace `Ruoqing Zhu (rqzhu)` by your name and NetID if you are using this template).
    - Make all of your `R` code chunks visible for grading.
    - Relevant outputs from your `R` code chunks that support your answers.
    - Provide clear answers or conclusions for each question. For example, you could start with `Answer: I fit SVM with the following choice of tuning parameters ...`
    - Many assignments require your own implementation of algorithms. **Basic comments are strongly encouraged** to explain the logic to our graders. However, line-by-line code comments are unnecessary.
- Requirements regarding the `.Rmd` file.
    - You do **NOT** need to submit `Rmd` files. However, your PDF file should be rendered directly from it.
    - Make sure that you **set random seeds** for simulation or randomized algorithms so that the results are reproducible. If a specific seed number is not provided in the homework, you can consider using your NetID.
    - 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.

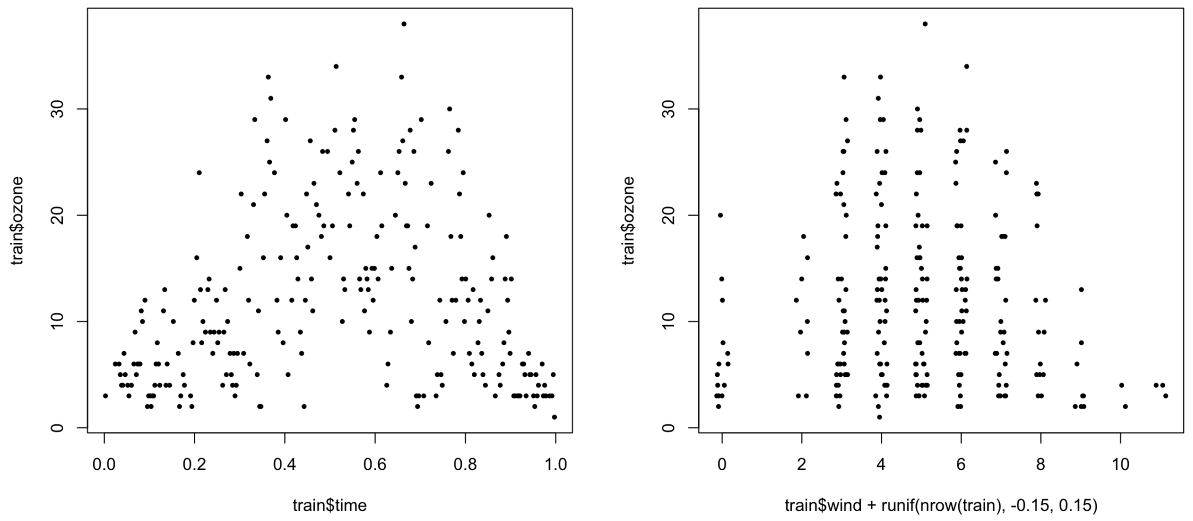
About HW6

- Understand how the spline basis is constructed. We will use the `Ozone` data from the `mlbench` package.
- Kernel regression involves two decisions: choosing the kernel and tuning the bandwidth. Usually, tuning the bandwidth is more influential than choosing the kernel function. Tuning the bandwidth is similar to tuning *k* in a KNN model. However, this is more difficult in multi-dimensional models. We practice one and two-dimensional kernels that involves these elements.

Question 1 [40 Points] Write Your Own Spline Basis (Univariate Spline Fit)

We will fit and compare different spline models to the `Ozone` dataset form the `mlbench` package. We have pre-processed this dataset. Please use our provided train/test csv files. This dataset has three variables: `time`, `ozone` and `wind`. For the spline question, we will only use `time` as the covariate and `ozone` as the outcome.

```
train = read.csv("ozoneTrain.csv")
test  = read.csv("ozoneTest.csv")
par(mfrow=c(1,2))

plot(train$time, train$ozone, pch = 19, cex = 0.5)
plot(train$wind + runif(nrow(train), -0.15, 0.15),
     train$ozone, pch = 19, cex = 0.5)
```



Let's consider several different spline constructions to model the `ozone` level using `time`. Please read the requirements carefully since some of them require you to write your own code. - To test your model, use the train/test split provided above. - Use the mean squared error as the metric for evaluation and report the MSE for each method in a single summary table. - For the spline basis that you write with your own code, make sure to include the intercept term. - For question a) and b) and d), provide the following summary information at the end of your answer: - A table of MSE on testing data. Please label each method clearly. - Three figures (you could consider making them side-by-side for a better comparison) at the end. Each figure consists of scatter plots of both training and testing data points and the fitted curve on the range `x = seq(0, 1, by = 0.01)`.

a. [10 pts] Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piece-wise linear fitting. Pick 4 knots at $(0.2, 0.4, 0.6, 0.8)$.

```
knots2=c(0.2, 0.4, 0.6, 0.8)
train_data=as.data.frame(cbind("time"=train$time,"ozone"=train$ozone))
test_data=as.data.frame(cbind("time"=test$time,"ozone"=test$ozone))


#linear spline
lin_spline<-function(data_x, knots){
      basis = cbind("int" = 1, "x_1" = data_x,
                    "x_2" = (data_x - knots[1])*((data_x - knots[1])>0),
                    "x_3" = (data_x - knots[2])*((data_x - knots[2])>0),
                    "x_4" = (data_x - knots[3])*((data_x - knots[3])>0),
                    "x_5" = (data_x - knots[4])*((data_x - knots[4])>0))
      basis

}

mse<-function(Y_pred, Y_real){
  mean((Y_real-Y_pred)^2)
}
```

b. [10 pts] Write your own code to implement a quadratic spline fitting. Your spline should be continuous up to the first derivative. Pick 4 knots as $(0.2, 0.4, 0.6, 0.8)$.

```
#quadratic spline
quad_spline<-function(data_x, knots){
      basis = cbind("int" = 1, "x_1" = data_x,
                    "x_2" = ((data_x - knots[1])^2)*((data_x - knots[1])>0),
                    "x_3" = ((data_x - knots[2])^2)*((data_x - knots[2])>0),
                    "x_4" = ((data_x - knots[3])^2)*((data_x - knots[3])>0),
                    "x_5" = ((data_x - knots[4])^2)*((data_x - knots[4])>0))
      basis
}
```

c. [10 pts] Produce a set of B-spline basis with the same knots and degrees as (ii) using the `bs()` function. Note that they do not have to be exactly the same as yours. Compare the design matrix from b) and c) as follows:

- Check the the difference between their projection matrices (the hat-matrices of the corresponding linear regression) on the training data to verify that the column spaces are the same. The difference is measured by $\max_{i,j} |H^{(1)}_{i,j} - H^{(2)}_{i,j}|$, where $H^{(1)}$ and $H^{(2)}$ are corresponding to the two hat-matrices.
- Compare the conditional number $\frac{\sigma_{\max}}{\sigma_{\min}}$ of each deign matrix, where $\sigma_{\max}$ and $\sigma_{\min}$ denote the largest and smallest singular values, respectively.
- [bonus 2 pts] Why B-spline has a smaller condition number even though two design matrices have the same column space. Some basic information of the conditional number (for linear regression) can be found here (https://en.wikipedia.org/wiki/Condition_number).

```
library(splines)
library(matlib)

#find design matrix
matrix_bs = bs(train$time, degree = 2, knots = knots2)
matrix_quad=quad_spline(train_data$time,knots2)

#Hat matrix

H_bs=matrix_bs%*%solve(t(matrix_bs)%*%(matrix_bs))%*%t(matrix_bs)
H_quad=matrix_quad%*%solve(t(matrix_quad)%*%(matrix_quad))%*%t(matrix_quad)

#calculate max of hat matrix dif
dif=max(abs(H_bs-H_quad))



#calculate conditional number
bs.condit=kappa(matrix_bs)
quad.condit=kappa(matrix_quad)

condit_numb=quad.condit/bs.condit

print(paste0("the difference between their projection matrices  = ",dif))
```

```
## [1] "the difference between their projection matrices  = 0.0595238961890076"
```

```
print(paste0("ratio of conditional numbers  = ",condit_numb))
```

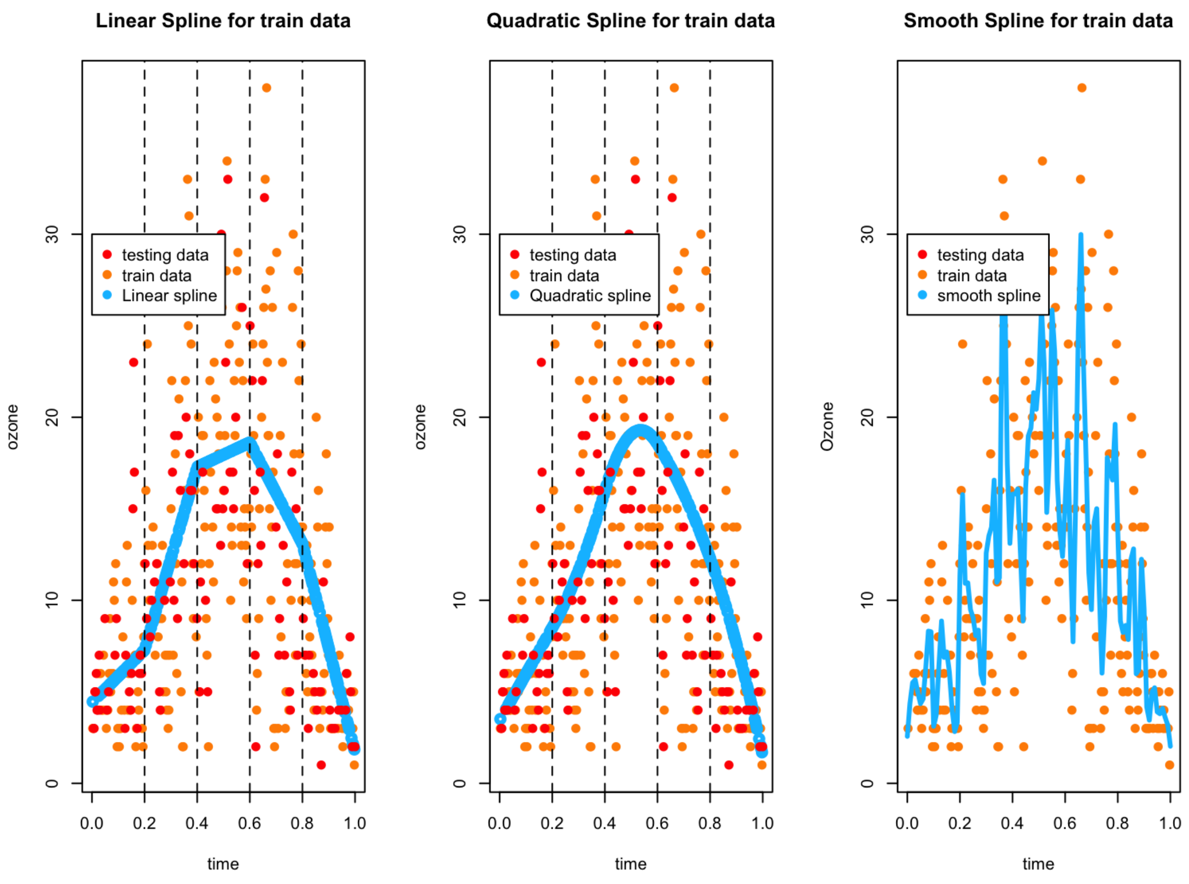
```
## [1] "ratio of conditional numbers  = 202.855349384175"
```

```
#bs have smaller conditional number
```

d. [10 pts] Use existing functions to implement a smoothing spline. Use the built-in generalized cross-validation method to select the best tuning parameter.

Summary for a,b and d

```
x = seq(0, 1, by = 0.01)

#mse function
mse<-function(Y_pred, Y_real){
  mean((Y_real-Y_pred)^2)
}

#plot 3 graphs for linear spline, quadratic spline and smooth spline
par(mfrow=c(1,3))

lf_train_1 <- lm(train_data$ozone ~ .-1, data = data.frame(lin_spline(train_data$ti
me,knots2)))

plot(train_data, pch = 19, col = "darkorange", xlab = "time", ylab = "ozone")
points(train_data$time, lf_train_1$fitted.values, lty = 1, col = "deepskyblue", lwd
= 3)
points(test_data$time, test_data$ozone,pch = 19, col="red")
abline(v = knots2, lty = 2)
legend(0, 30,legend=c("testing data","train data", "Linear spline"), col=c("red","d
arkorange", "deepskyblue"),pch=c(19,19,19), ncol=1)
title("Linear Spline for train data")

lf_train_2 <- lm(train_data$ozone ~ .-1, data = data.frame(quad_spline(train_data$t
ime,knots2)))
plot(train_data, pch = 19, col = "darkorange",xlab = "time", ylab = "ozone")
    points(train_data$time, lf_train_2$fitted.values, lty = 1, col = "deepskyblue",
lwd = 3)
    points(test_data$time, test_data$ozone,pch = 19, col="red")
    abline(v = knots2, lty = 2)
    legend(0, 30,legend=c("testing data","train data", "Quadratic spline"), col=c(
"red","darkorange", "deepskyblue"),pch=c(19,19,19), ncol=1)
    title("Quadratic Spline for train data")

fit = smooth.spline(train$time, train$ozone)
    plot(train$time, train$ozone, pch = 19,
         xlab = "time", ylab = "Ozone", col = "darkorange")
    lines(seq(0, 1, by = 0.01), predict(fit, seq(0, 1, by = 0.01))$y, col="deepskyb
lue", lty=1, lwd = 3)
    legend(0, 30,legend=c("testing data","train data", "smooth spline"), col=c("re
d","darkorange", "deepskyblue"),pch=c(19,19,19), ncol=1)
    title("Smooth Spline for train data")
```



```
#Report fit$lambda for smooth spline
print(paste0("the best lambda  = ",fit$lambda))
```

```
## [1] "the best lambda  = 1.24299664679981e-07"
```

```
#Calculate and report mse for linear spline, quadratic spline and smooth spline
y_lin=predict(lf_train_1, newdata =data.frame(lin_spline(test_data$time,knots2)))
y_quad=predict(lf_train_2, newdata =data.frame(quad_spline(test_data$time,knots2)))

mse_lin=mse(y_lin, test_data$ozone)
mse_quad=mse(y_quad, test_data$ozone)
mse_smooth_spline=mse(predict(fit, test_data$time)$y, test_data$ozone)

#create table of MSE from different methods
mse<-matrix(c(mse_lin, mse_quad,mse_smooth_spline), ncol = 3, byrow = TRUE)
mse_table<-as.data.frame(mse)
colnames(mse_table)<-c("MSE for linear spline", "MSE for quadratic spline", "MSE fo
r smooth spline")


mse_table
```

MSE for linear spline <dbl>	MSE for quadratic spline <dbl>	MSE for smooth spline <dbl>
31.08565	29.81354	23.71095

1 row

Question 2 [60 Points] Kernel Regression

We will use the same ozone data. For Question a), we only use `time` as the covariate, while in Question b, we use both `time` and `wind`.

a) [30 pts] One-dimensional kernel regression.

You are required to implement (write your own code) two kernel regression models, using the following two kernels function, respectively:

- Gaussian kernel, defined as $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$
- Epanechnikov kernel, defined as $K(u) = \frac{3}{4}(1 - u^2)$ for $|u| \leq 1$.

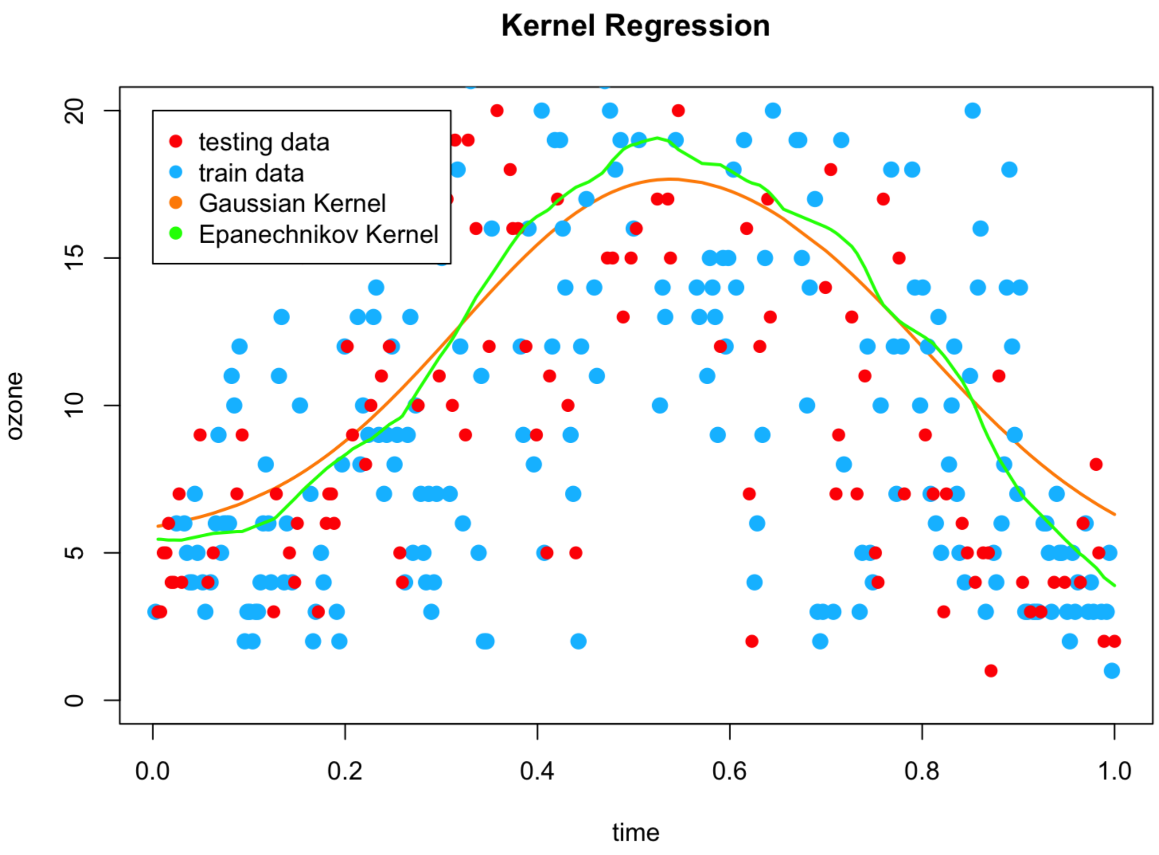
For both kernel functions, incorporate a bandwidth λ. You should start with the Silverman's rule-of-thumb for the choice of λ, and then tune λ (for example, increase or decrease by 10%, 20%, 30% etc.). Then, perform the following:

- I. [20 pts] Using just the Silverman's rule-of-thumb, fit and plot the regression line with both kernel functions, in a single figure. Add the training/testing points, just like Question 1. Report the testing MSE of both methods in a table.
- II. [10 pts] For the Epanechnikov kernel, tune the λ value by minimizing the testing error. Use a grid of 10 different λ values at your choice. What is the best λ value that minimizes the testing error? Plot your optimal regression line and report the best λ and the testing error.

part 1

```
#gaussian kernel function
pi=3.141593
gaussian<-function(x){
  (1/(pi^0.5))*exp(-(x^2)/2)
}

#calculate Y_pred from gaussian kernel
gaussian_pred<-function(x,x1,y1,lambda){
  y_pred=rep(0,length(x))
  for(j in 1:length(x)){
    numer=0
    denom=0
     for(i in 1:length(x1)){
       x_dif=(abs(x[j]-x1[i])/lambda)
       numer=numer+(gaussian(x_dif)*y1[i])/lambda
       denom=denom+(gaussian(x_dif))/lambda
     }
     y_pred[j]=numer/denom

  }
   y_pred
}

#epanechnikov kernel
epanechnikov<-function(x){
  if(abs(x)>1){
    y=0
  }else{
    y=0.75*(1-x^2)
  }
  y
}

#calculate Y_pred from epanechnikov kernel
epanechnikov_pred<-function(x,x1,y1,lambda){
  y_pred=rep(0,length(x))
  for(j in 1:length(x)){
    numer=0
    denom=0
     for(i in 1:length(x1)){
       x_dif=(abs(x[j]-x1[i])/lambda)
       numer=numer+(epanechnikov(x_dif)*y1[i])/lambda
       denom=denom+(epanechnikov(x_dif))/lambda
     }
     y_pred[j]=numer/denom
  }
   y_pred
}

#mse function
```

```r
mse<-function(Y_pred, Y_real){
  mean((Y_real-Y_pred)^2)
}

#calculate lambda using silverman rule
silverman_lambda = 1.06*sd(train$time)*length(train$time)^(-1/5)

#fing y_pred from 2 kernela
y_gaus=gaussian_pred(test$time,train$time,train$ozone,silverman_lambda)
y_epanechnikov=epanechnikov_pred(test$time,train$time,train$ozone,silverman_lambda)

#plot graphs
plot(test$time, y_gaus, ylim=c(0,20),col= "darkorange", type="l", lwd=2, xlab = "ti
me", ylab = "ozone")
points(train_data$time, train_data$ozone, pch=19, col = "deepskyblue", lwd = 3)
points(test_data$time, test_data$ozone,pch = 19, col="red")
lines(test$time, y_epanechnikov, col="green", lwd = 2)
legend(0, 20,legend=c("testing data","train data", "Gaussian Kernel", "Epanechnikov
Kernel"), col=c("red","deepskyblue", "darkorange","green"),
                                   pch=c(19,19,19,19), ncol=1)
title("Kernel Regression")
```



```r
#calculate mse for 2 kernels and report them
mse_gaus=mse(y_gaus, test_data$ozone)
mse_epan=mse(y_epanechnikov, test_data$ozone)

#create table of MSE for different methods
mse2<-matrix(c(mse_gaus, mse_epan), ncol = 2, byrow = TRUE)
mse_table2<-as.data.frame(mse2)
colnames(mse_table2)<-c("MSE for Gaussian Kernel", "MSE for Epanechnikov kernel")
mse_table2
```

MSE for Gaussian Kernel <dbl>	MSE for Epanechnikov kernel <dbl>
30.07586	29.87374

1 row

part 2

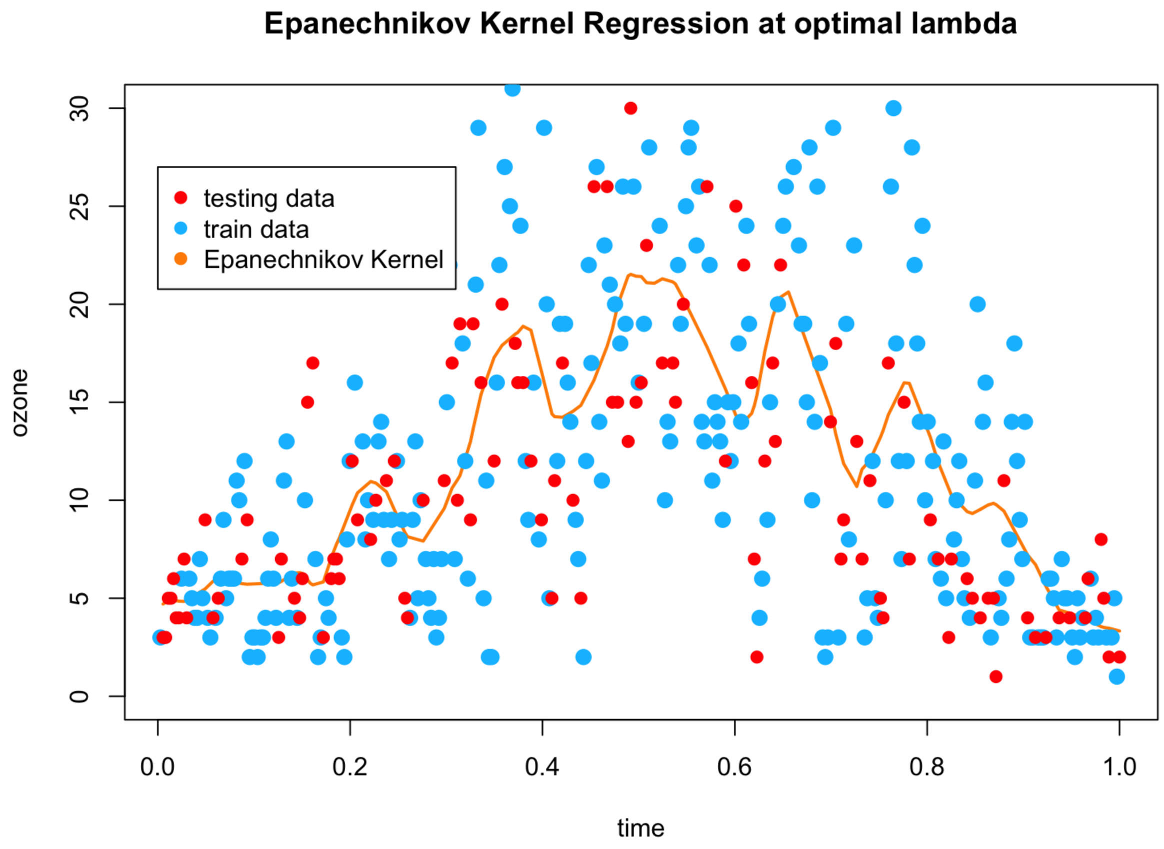
```r
#part 2
# create 10 lambda by adding and substracting 10%
lambda=c(1.4,1.3,1.2,1.1,0.9,0.8,0.7,0.6,0.5,0.4)
lambda_new=silverman_lambda*lambda

#fit epanechnikov kernel model and calculate mse for 10 lambdas
mse_lambda=rep(0,10)

for(i in 1:10){
  y_epanechnikov2=epanechnikov_pred(test$time,train$time,train$ozone,lambda_new[i])
  mse_lambda[i]=mse(y_epanechnikov2, test_data$ozone)
}

#find minimum mse and corresponding optimal lamba
mse_min=min(mse_lambda)
lambda_optim=lambda_new[which(mse_lambda == mse_min)]
print(paste0("Optimal lambda  = ",lambda_optim))
```

```
## [1] "Optimal lambda  = 0.0406121392909835"
```

```r
print(paste0("The corresponding minimum MSE at lambda optimal  = ",mse_min))
```

```
## [1] "The corresponding minimum MSE at lambda optimal  = 27.8202698510814"
```

```
#fit epanechnikov kernel model with optimal lambda and plot graph
y_optim=epanechnikov_pred(test$time,train$time,train$ozone,lambda_optim)
plot(test$time, y_optim, ylim=c(0,30),col= "darkorange", type="l", lwd=2, xlab = "t
ime", ylab = "ozone")
points(train_data$time, train_data$ozone, pch=19, col = "deepskyblue", lwd = 3)
points(test_data$time, test_data$ozone,pch = 19, col="red")
legend(0, 27,legend=c("testing data","train data","Epanechnikov Kernel"), col=c("re
d","deepskyblue", "darkorange"),pch=c(19,19,19), ncol=1)
title("Epanechnikov Kernel Regression at optimal lambda")
```



b [25 Points] Two-dimensional Kernel

We consider using both `time` and `wind` in the regression. We use the following multivariate kernel function, which is essentially a Gaussian kernel with diagonal covariance matrix. You can also view this as the product of two kernel functions, corresponding to the two variables:

$$K_\lambda(x, z) \propto \exp\left\{ -\frac{1}{2} \sum_{j=1}^{p} \left((x_j - z_j)/\lambda_j \right)^2 \right\}$$

Based on the Silverman's formula, the bandwidth for the jth variable is given by

$$\lambda_k = \left(\frac{4}{p+2} \right)^{\frac{1}{p+4}} n^{-\frac{1}{p+4}} \, \hat{\sigma}_j,$$

where $\hat{\sigma}_j$ is the estimated standard deviation for variable j. Use the Nadaraya-Watson kernel estimator to fit and predict the `ozone` level using `time` and `wind`. At the end, report the testing error.

```r
train_cov=as.data.frame(cbind("time"=train$time, "wind"=train$wind))
test_cov=as.data.frame(cbind("time"=test$time, "wind"=test$wind))

#function for lambda
multivar_lambda<-function(covar,n,p){
  ((4/(p+2))^(1/(p+4)))*(n^(-1/(p+4)))*sd(covar)
}

#calculate 2 lamvdas
lambda=rep(0,2)
lambda[1]=multivar_lambda(train_cov$time, nrow(train_cov), ncol(train_cov))
lambda[2]=multivar_lambda(train_cov$wind, nrow(train_cov), ncol(train_cov))

#kernel function
kernel<-function(x,y, lambda){
  exp(-0.5*(((x-y)/lambda)^2))
}




# find y_pred for 2 covariate model
y_pred=rep(0,nrow(test_cov))
  for(j in 1:nrow(test_cov)){
    numer=0
    denom=0
     for(i in 1:nrow(train_cov)){
       numer=numer+kernel(test_cov$time[j],train_cov$time[i],lambda[1])*kernel(test
_cov$wind[j],train_cov$wind[i], lambda[2])*train$ozone[i]
       denom=denom+kernel(test_cov$time[j],train_cov$time[i],lambda[1])*kernel(test
_cov$wind[j],train_cov$wind[i], lambda[2])
     }

     y_pred[j]=numer/denom
  }

#calculate mse
mse<-function(Y_pred, Y_real){
  mean((Y_real-Y_pred)^2)
}

mse_multiv=mse(y_pred, test_data$ozone)

print(paste0("The testing error = ",mse_multiv))
```

```
## [1] "The testing error = 32.38575692246"
```

c [5 Points] Variance of Multi-dimensional Kernel

In our lecture, we only introduced the one-dimensional case for density estimations. For a regression problem, the rate is essentially the same. However, when we have multivariate kernels, the rate of bias and variance will change. If we use the same λ from the one-dimensional case in a two-dimensional problem, would you expect the bias to increase/decrease/unchanged? Would you expect the variance to increase/decrease/unchanged? And why? Hint: for the same λ, bias is quantified by how far your neighboring points are, and variance is quantified by how many points you are capturing within that neighborhood.

Answer

> When we use the same lambda, it becomes similar to KNN model. Distance between points increased in high dimensional case, so bias will increases as bias is quantified by how far your neighboring points are, and variance will decrease as variance is quantified by how many points you are capturing within that neighborhood.