STAT 542: Homework 4

Spring 2020, by Yifan Shi (yifans16)

Due: Friday, Mar 20, by 11:59 PM

Contents

Question 1 [40 Points] K-Means Clustering

Let's consider coding our own K-means algorithm. Perform the following:

- [15 Points] Write your own code of k-means that iterates between two steps, and stop when the cluster membership does not change. Use the ℓ_2 norm as the distance metric. Write your algorithm as efficiently as possible. Avoiding extensive use of for-loop could help.
  - updating the cluster means given the cluster membership
  - updating the cluster membership based on the cluster means

```
my_kmeans <- function(data,k){
  #randomly initialize centers
  #data_min = apply(data,2,min)
  #data_max = apply(data,2,max)
  #centers = matrix(runif(k*ncol(data),data_min,data_max),
  #                 k,ncol(data),byrow=T)
  centers = data[sample(nrow(data),k,replace=F),]
  #1st col is the min dist
  #2nd/3rd cols are the new/old assigned centers
  summary = matrix(c(Inf,0,0),nrow(data),3,byrow=T)
  diff = T
  while (diff== T){
    diff = F
    #determine the registration of cluster for each obs
    for (i in 1:nrow(data)){
      dis = as.numeric(as.matrix(dist(rbind(data[i,],centers)))[-1,1])
      summary[i,1] = min(dis)
      summary[i,2] = which.min(dis)
    }
    if (any(summary[,2] != summary[,3])){
      diff = T
      summary[,3] = summary[,2]
      #update the centers by means of clusters
      for (j in 1:k){
        centers[j,] = apply(data[which(summary[,2]==j),],2,mean)
      }
    }
  }
  result = list(summary[,1],summary[,2]-1,centers)
  return(result)
}
```

- [5 points] Test your code with this small-sized example by performing the following. Make sure that you save the random seed.
  - Run your algorithm on the data using **ONE** random initialization, output the within-cluster distance, and compare the resulting clusters with the underlying truth.
  - Run your algorithm on the data using **20** random initialization, output the within-cluster distance, and compare the resulting clusters with the underlying truth.
  - Do you observe any difference? Note that the result may vary depending on the random seed.

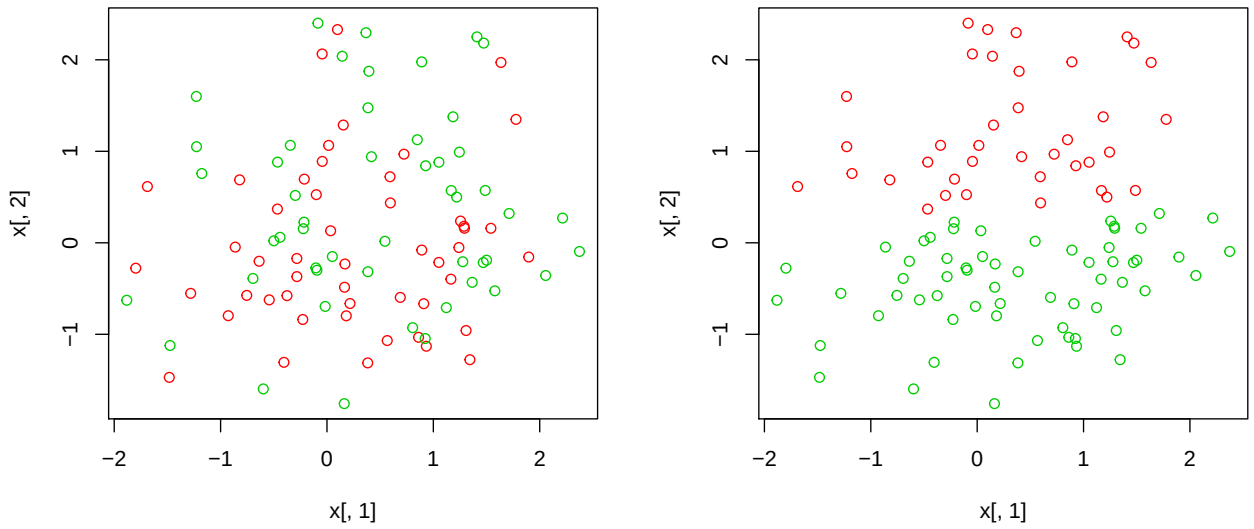
```
  set.seed(4)
  x = rbind(matrix(rnorm(100), 50, 2), matrix(rnorm(100, mean = 0.5), 50, 2))
  y = c(rep(0, 50), rep(1, 50))
```

The within-cluster distance is 99.48548

```
set.seed(542)
clus1 = my_kmeans(x,2)
sum(clus1[[1]])
```

```
## [1] 99.48548
```

```
par(mfrow = c(1,2))
plot(x[,1],x[,2],col=y+2)
plot(x[,1],x[,2],col=clus1[[2]]+2)
```



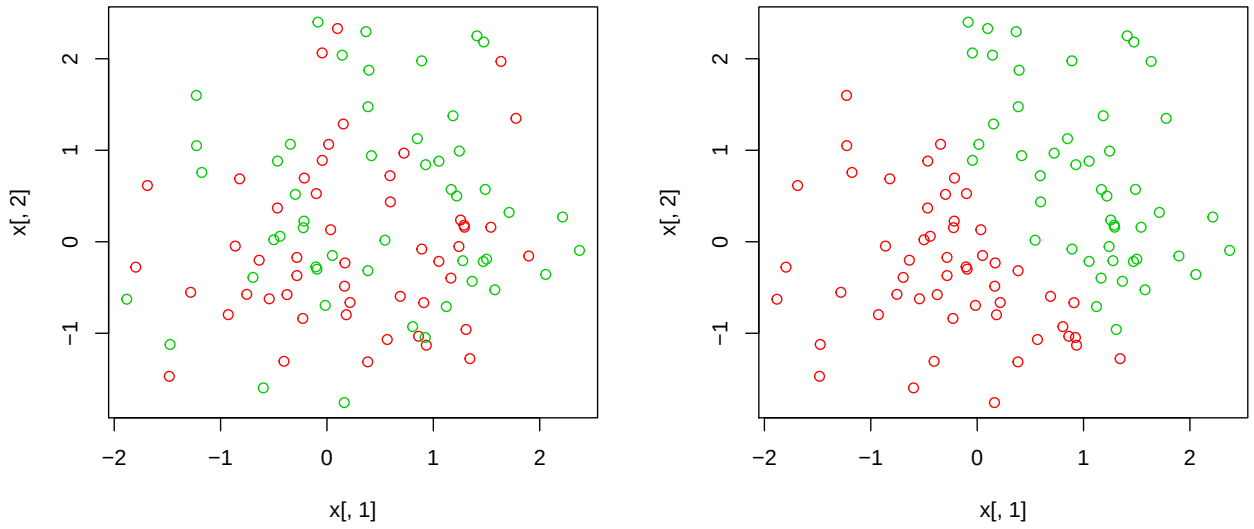

```
set.seed(542)
clus_dist1 = matrix(0,20,nrow(x))
clus_label1 = matrix(0,20,length(y))
for (a in 1:20) {
  temp_list = my_kmeans(x,2)
  clus_dist1[a,] = temp_list[[1]]
  clus_label1[a,] = temp_list[[2]]
}
```

The within-cluster distance is 95.72488

```
clus2 = clus_label1[which.min(apply(clus_dist1,1,sum)),]
min(apply(clus_dist1,1,sum))
```

```
## [1] 95.72488
```

```
par(mfrow = c(1,2))
plot(x[,1],x[,2],col=y+2)
plot(x[,1],x[,2],col=clus2+2)
```



- [10 Points] Load the `zip.train` (handwritten digit recognition) data from the `ElemStatLearn` package and the goal is to identify clusters of digits based on only the pixels. Apply your algorithm on the handwritten digit data with $k = 15$, with ten random initialization. Then, compare your cluster membership to the true digits.
  - For each of your identified clusters, which is the representative (most dominating) digit?
  - How well does your clustering result recover the truth? Which digits seem to get mixed up (if any) the most? Again, this may vary depending on the random seed. Hence, make sure that you save the random seed.

```
library(ElemStatLearn)
dat_trn <- zip.train
dat_tst <- zip.test
trn.x <- dat_trn[,2:257]
trn.y <- dat_trn[,1]
tst.x <- dat_tst[,2:257]
tst.y <- dat_tst[,1]
```

```
set.seed(542)
clus_dist2 = matrix(0,10,nrow(trn.x))
clus_label2 = matrix(0,10,length(trn.y))
clus_center2 = list()
for (b in 1:10) {
  temp_list = my_kmeans(trn.x,15)
  clus_dist2[b,] = temp_list[[1]]
  clus_label2[b,] = temp_list[[2]]
  clus_center2[[b]] = temp_list[[3]]
}
```

```
clus = clus_label2[which.min(apply(clus_dist2,1,sum)),]
#make a matrix to check freq of each cluster
temp = matrix(0,15,10)
for (c in 1:length(clus)) {
  temp[clus[c]+1,trn.y[c]+1] = temp[clus[c]+1,trn.y[c]+1] + 1
```

```
}
#make a dictionary of digits for the clusters
dict = rep(0,15)
for (d in 1:nrow(temp)) {
  dict[d] = which.max(temp[d,])
}
#update the clusters with their corresponding digits
for (e in 1:length(clus)) {
  clus[e] = dict[clus[e]+1] - 1
}
```

The dominating digit for each cluster is:

```
dictionary = data.frame("Cluster"=seq(1,15),"Digit"=dict-1)
knitr::kable(dictionary)
```

Cluster	Digit
1	2
2	0
3	4
4	5
5	9
6	2
7	3
8	0
9	6
10	8
11	0
12	0
13	1
14	1
15	7

```
sum(as.numeric(clus==trn.y))/length(trn.y)
```

```
## [1] 0.7842546
```

The accuracy is 78.4%

```
mix = rep(0,10)
for (j in 1:10) {
  mix[j] = mean(clus[which(trn.y==j-1)]!=trn.y[which(trn.y==j-1)])
}
knitr::kable(data.frame("Digit"=seq(0,9),"Error Rate"=mix))
```

Digit	Error.Rate
0	0.0770519
1	0.0009950
2	0.1682627
3	0.1610942
4	0.5276074
5	0.2859712
6	0.3057229

Digit	Error.Rate
7	0.3240310
8	0.2380074
9	0.3214286

Digit 4 gets mixed up the most.

- [10 Points] If you are asked to use this clustering result as a predictive model and suggest (predict) a label for each of the testing data `zip.test`, what would you do? Properly describe your prediction procedure and carry out the analysis. What is the prediction accuracy of your model? Note that you can use the training data labels for prediction.

I would use the cluster centers that calculated from the training set and determine which cluster is the closest one for each obs in the testing set. Then I will use the dictionary of the digits for the clusters to determine the predcition of digits for each obs.

```
clus_center = clus_center2[[which.min(apply(clus_dist2,1,sum))]]
clus_tst = rep(0,length(tst.y))
for (f in 1:nrow(tst.x)) {
  dis = as.numeric(as.matrix(dist(rbind(tst.x[f,],clus_center)))[-1,1])
  clus_tst[f] = dict[which.min(dis)] - 1
}
sum(as.numeric(clus_tst==tst.y))/length(tst.y)
```

```
## [1] 0.7518685
```

The prediction accuracy is 75.2%

Question 2 [20 Points] Kernel Density Estimation

If you do not use latex to type your answer, you will lose 2 points. During our lecture, we analyzed the consistency of a kernel density estimator, defined as

$$\widehat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x, x_i)$$

Let's consider a 2-dimensional case, where $x = (z, w)^T$, and $x_i = (z_i, w_i)^T$. If we use a 2-dimensional kernel, constructed as the product of two kernel functions:

$$\mathbf{K}_\lambda(x, x_i) = K_\lambda(z, z_i)K_\lambda(w, w_i)$$

We would expect our kernel density estimation still being an unbiased estimator. Show this by extending the one-dimensional proof we did during the lecture to this new case. Please be aware of the following and make sure that you properly address them during the proof.

- In the one-dimensional proof, we require assumptions on the kernel function $K(\cdot)$. You can automatically assume these in your proof.
- A two-dimensional Taylor expansion is needed.
- Is this kernel density estimator consistent?
- What is the rate (in terms of λ) of the bias term following your derivation? Is the rate slower or faster than the one-dimensional case?
- What do you think about the variance of this estimator? You do NOT need to give proof. Providing some thoughts is sufficient. This question does not count towards your grade.

The assumptions:

$$K(-t) = K(t) \int K(t)dt = 1 \int K(t)tdt = 0 \int K(t)t^2dt < \infty$$

Then the kernel estimated density function can be derived as:

$$\begin{aligned} f(x) = f(z,w) &= \mathbb{E}(\hat{f}(z,w)) \\ &= \mathbb{E}(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda}K(\frac{z-z_i}{\lambda})\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda}K(\frac{w-w_i}{\lambda})) \\ &= \mathbb{E}(\frac{1}{\lambda^2}K(\frac{z-z_1}{\lambda})K(\frac{w-w_1}{\lambda})) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{1}{\lambda^2}K(\frac{z-z_1}{\lambda})K(\frac{w-w_1}{\lambda})f(z_1,w_1)dz_1dw_1 \end{aligned}$$

Let $a = \frac{z-z_1}{\lambda}$ and $b = \frac{w-w_1}{\lambda}$ then $z_1 = z - \lambda a$ and $w_1 = w - \lambda b$

$$\begin{aligned} f(x) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{1}{\lambda^2}K(a)K(b)f(z-\lambda a, w-\lambda b)d(z-\lambda a)d(w-\lambda b) \\ &= \iint_{\mathbb{R}^{\not\varkappa}} K(a)K(b)f(z-\lambda a, w-\lambda b)\ d(a)d(b) \\ &= \iint_{\mathbb{R}^{\not\varkappa}} K(a)K(b)[f(z,w) - \begin{bmatrix}\lambda a \\ \lambda b\end{bmatrix}\nabla f(z,w) + \frac{1}{2}[\lambda a\ \lambda b]\mathbb{H}(z,w)\begin{bmatrix}\lambda a \\ \lambda b\end{bmatrix} + o(\lambda^2)]d(a)d(b) \\ &= f(z,w) - 0 + \frac{\lambda^2}{2}\iint_{\mathbb{R}^{\not\varkappa}} K(a)K(b)[a\ b]\mathbb{H}(z,w)\begin{bmatrix}a \\ b\end{bmatrix}d(a)d(b) + o(\lambda^2) \\ &= f(z,w) \qquad (\text{as } \lambda \to 0) \end{aligned}$$

Thus this kernel density estimator is consistent.

Bias can now be derived as:

$$\begin{aligned} \text{Bias} &= \mathbb{E}[\hat{f}(z,w)] - f(z,w) \\ &= f(z,w) + \frac{\lambda^2}{2}\iint_{\mathbb{R}^{\not\varkappa}} K(a)K(b)[a\ b]\mathbb{H}(z,w)\begin{bmatrix}a \\ b\end{bmatrix}d(a)d(b) + o(\lambda^2) - f(z,w) \\ &= \frac{\lambda^2}{2}\iint_{\mathbb{R}^2} K(a)K(b)[\frac{\partial^2 f}{\partial z^2}(a)^2 + \frac{\partial^2 f}{\partial w^2}(b)^2 + 2\frac{\partial^2 f}{\partial z \partial w}(ab)]d(a)d(b) + o(\lambda^2) \\ &= \frac{\lambda^2}{2}\int_a [\frac{\partial^2 f}{\partial z^2}a^2K(a) + \frac{\partial^2 f}{\partial w^2}\sigma_K^2 K(b)]da + o(\lambda^2) \\ &= \frac{\lambda^2}{2}[\frac{\partial^2 f}{\partial z^2}\sigma_K^2 + \frac{\partial^2 f}{\partial w^2}\sigma_K^2] + o(\lambda^2) \\ &= \frac{\lambda^2\sigma_K^2}{2}[\frac{\partial^2 f}{\partial z^2} + \frac{\partial^2 f}{\partial w^2}] + o(\lambda^2) \end{aligned}$$

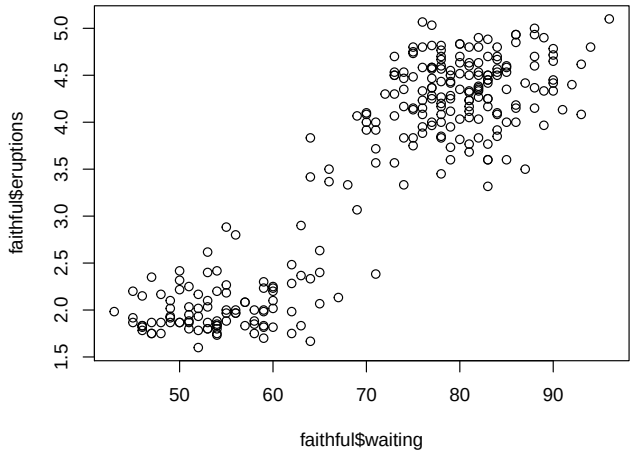
The rate of bias does not change which is still λ^2

$$\text{Var} \approx \frac{1}{n\lambda} \iint_{\mathbb{R}^2} K^2(a)K^2(b)d(a)d(b)$$

Question 3 [40 Points] Two-dimensional Kernel Regression

For this question, you need to write your own functions. The goal is to model and predict the eruption duration using the waiting time in between eruptions of the Old Faithful Geyser. You can download the data from [kaggle: Old Faithful] or our course website.

```
load("faithful.Rda")
plot(faithful$waiting, faithful$eruptions)
```

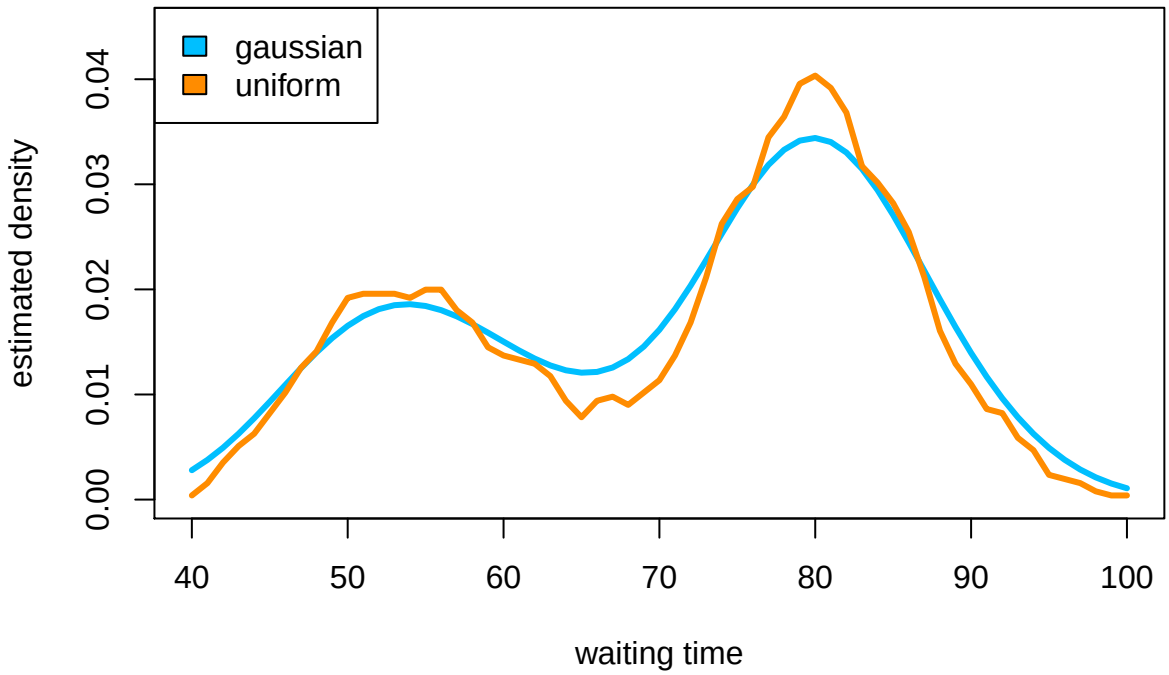


Perform the following:

- [10 points] Use a kernel density estimator to estimate the density of the eruption waiting time. Use a bandwidth with the theoretical optimal rate (the Silverman rule-of-thumb formula). Plot the estimated density function. You should consider two kernel functions:
  - Gaussian Kernel
  - Uniform kernel

```
newx = seq(40,100,1)
x = faithful$waiting
y = faithful$eruptions
bw = sd(x)*(4/3/length(x))^(1/5)
den1 = matrix(NA, length(x), length(newx))
den2 = matrix(NA, length(x), length(newx))
for (g in 1:length(x)) {
  den1[g,] = exp(-0.5*(x[g] - newx)^2 / bw^2)/sqrt(2*pi)/bw/length(x)
  den2[g,] = (abs(newx - x[g]) <= bw)/length(x)/2/bw
}
```

```
plot(newx, colSums(den1), type = "l", lwd = 3, col = "deepskyblue",
     ylim=c(0,0.045),xlab="waiting time",ylab="estimated density")
lines(newx, colSums(den2), lwd = 3, col = "darkorange")
legend("topleft",c("gaussian","uniform"),fill=c("deepskyblue","darkorange"))
```



- [15 points] Use a Nadaraya-Watson kernel regression estimator to model the conditional mean eruption duration using the waiting time. Use a 10-fold cross-validation to select the optimal bandwidth. You should use the Epanechnikov kernel for this task. Report the optimal bandwidth and plot the fitted regression line using the optimal bandwidth.

The formula of Epanechnikov kernel is:

$$\frac{0.75}{\lambda}(1-(\frac{x_i-x}{\lambda})^2)\ \mathbf{1}(|\frac{x_i-x}{\lambda}|<=1)$$

```
kernel_reg <- function(xtest,xtrain,ytrain,bw){
  #initialize ytest
  ytest <- rep(0,length(xtest))
  for (i in 1:length(xtest)){
    #the dist between each obs in tst and all obs in trn
    xi = rep(xtest[i],length(xtrain))-xtrain
    u = abs(xi/bw)
    u[u<=1] = 1
    u[u>1] = 0
    #calculate the Epanechnikov kernel
    xk = 0.75/bw*(1-(xi/bw)^2)*u
    ytest[i] = sum(ytrain*xk)/sum(xk)
  }
  return(ytest)
}
```

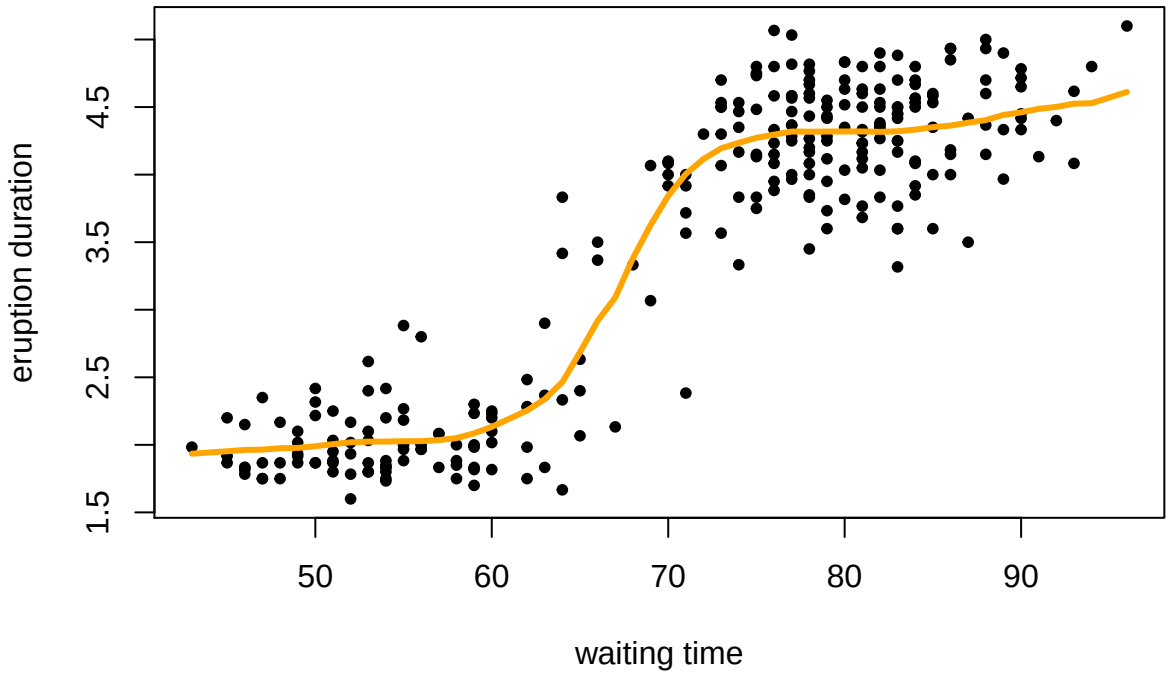
```
folds <- rep_len(1:10,length(x))
bw <-  seq(2.5,10,by=0.5)
MSE1 <-  matrix(NA,length(bw),10)
for (h in 1:length(bw)){
  for (i in 1:10){
    tst_index = which(folds==i)
    x_trn = x[-tst_index]
```

```
    y_trn = y[-tst_index]
    x_tst = x[tst_index]
    y_tst = y[tst_index]
    ypred = kernel_reg(x_tst,x_trn,y_trn,bw[h])
    MSE1[h,i] = sum((ypred-y_tst)^2)/length(y_tst)
  }
}
bw_opt1 = bw[which.min(apply(MSE1,1,mean))]
bw_opt1
```

```
## [1] 6
```

The optimal bandwidth is 6

```
plot(x,y,xlab="waiting time",ylab="eruption duration",pch=20)
lines(sort(x),kernel_reg(sort(x),x,y,bw_opt1),col="orange",lwd=3)
```



- [15 points] Fit a local linear regression to model the conditional mean eruption duration using the waiting time. Use a 10-fold cross-validation to select the optimal bandwidth. You should use the Gaussian kernel for this task. Report the optimal bandwidth and plot the fitted regression line using the optimal bandwidth.

```
local_reg <- function(xtest,xtrain,ytrain,bw){
  #initialize ytest
  ytest <- rep(0,length(xtest))
  for (i in 1:length(xtest)) {
    x0 = xtest[i]
    xi = rep(xtest[i],length(xtrain))-xtrain
    #the diagonal matrix of gaussian kernels
    xw = diag(1/(bw*sqrt(2*pi)) * exp(-(xi)^2/(2*bw^2)))
    #design matrix is b
    b = matrix(c(rep(1,length(xtrain)),xtrain),length(xtrain),2)
    inv = solve(t(b)%*%xw%*%b)
    newy = matrix(ytrain,length(ytrain),1)
```

```
    ytest[i] = matrix(c(1,x0),1,2)%*%inv%*%t(b)%*%xw%*%newy
  }
  return(ytest)
}
```

```
MSE2 <-  matrix(NA,length(bw),10)
for (h in 1:length(bw)){
  for (i in 1:10){
    tst_index = which(folds==i)
    x_trn = x[-tst_index]
    y_trn = y[-tst_index]
    x_tst = x[tst_index]
    y_tst = y[tst_index]
    ypred = local_reg(x_tst,x_trn,y_trn,bw[h])
    MSE2[h,i] = sum((ypred-y_tst)^2)/length(y_tst)
  }
}
bw_opt2 = bw[which.min(apply(MSE2,1,mean))]
bw_opt2
```

```
## [1] 3
```

The optimal bandwidth is 3

```
plot(x,y,xlab="waiting time",ylab="eruption duration",pch=20)
lines(sort(x),local_reg(sort(x),x,y,bw_opt2),col="orange",lwd=3)
```