STAT 527- Assignment 3

Joseph David

May 20 2022

Problem 1.

Proof. (a) First we define $\Psi \in \mathbb{R}^{n\times J}$ by

$$\Psi_{ij} = \psi_j(x_i).$$

Then, we see that

$$(\Psi\beta)_i = \sum_{j=1}^{J} \beta_j \psi_j(x_i),$$

so $(\underline{y} - \Psi\beta)_i = y_i - \sum_{j=1}^{J} \beta_j\psi_j(x_i)$ which shows that

$$||\underline{y} - \Psi\beta||_2^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} \beta_j \psi_j(x_i) \right)^2.$$

We also see that

$$\begin{aligned}\int_0^1 \left[\frac{\partial^2}{\partial x^2} \left(\sum_{j=1}^{J} \beta_j \psi_j(x) \right) \right]^2 dx &= \int_0^1 \left[\sum_{j=1}^{J} \beta_j \psi_j''(x) \right]^2 dx \\ &= \int_0^1 \sum_{j,k=1}^{J} \beta_j \psi_j''(x) \cdot \beta_k \psi_k''(x)\, dx.\end{aligned}$$

Define $\Omega \in \mathbb{R}^{J\times J}$ by $\Omega_{ij} = \int_0^1 \psi_i''(x)\psi_j''(x)\, dx$. We see that

$$\begin{aligned}(\Omega\beta)_i &= \sum_{j=1}^{J} \beta_j \int_0^1 \psi_i''(x)\psi_j''(x)\, dx \\ &= \int_0^1 \beta_j \psi_i''(x) \left(\sum_{j=1}^{J} \psi_j''(x) \right) dx,\end{aligned}$$

and therefore

$$\begin{aligned}\beta^T\Omega\beta &= \sum_{i=1}^{J} \beta_i \left(\sum_{j=1}^{J} \beta_j \int_0^1 \psi_i''(x)\psi_j''(x)\, dx \right) \\ &= \int_0^1 \sum_{j,k=1}^{J} \beta_j \psi_j''(x) \cdot \beta_k \psi_k''(x)\, dx \\ &= \int_0^1 \left[\frac{\partial^2}{\partial x^2} \left(\sum_{j=1}^{J} \beta_j \psi_j(x) \right) \right]^2 dx.\end{aligned}$$

This shows that

$$||\underline{y} - \Psi\beta||_2^2 + \lambda\beta^T\Omega\beta = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{J}\beta_j\psi_j(x_i)\right)^2 + \lambda\int_0^1\left[\frac{\partial^2}{\partial x^2}\left(\sum_{j=1}^{J}\beta_j\psi_j(x)\right)\right]^2 dx,$$

which completes the problem. □

Proof. (b) Using the result from (a), we find $\widehat{\beta}$ by differentiating the loss function with respect to β, set it equal to zero and solve.

We see that

$$\frac{\partial}{\partial\beta}||\underline{y} - \Psi\beta||_2^2 = -2\Psi^T(\underline{y} - \Psi\beta)$$

$$\frac{\partial}{\partial\beta}\lambda\beta^T\Omega\beta = 2\lambda\Omega\beta,$$

and thus we solve

$$\begin{aligned}
&-2\Psi^T(\underline{y} - \Psi\widehat{\beta}) + 2\lambda\Omega\widehat{\beta} = 0\\
&\Rightarrow \Psi^T\underline{y} - \Psi^T\Psi\widehat{\beta} - \lambda\Omega\widehat{\beta} = 0\\
&\Rightarrow \Psi^T\underline{y} = (\Psi^T\Psi + \lambda\Omega)\widehat{\beta}\\
&\Rightarrow \widehat{\beta} = (\Psi^T\Psi + \lambda\Omega)^{-1}\Psi^T\underline{y}.
\end{aligned}$$

□

Proof. (c) We use the penalization parameter λ to control the bias-variance tradeoff and thus only need to cross validate λ and not number of basis functions J. Thus, it does not matter if we choose a large J since a higher choice of penalization parameter will smooth our regression function.

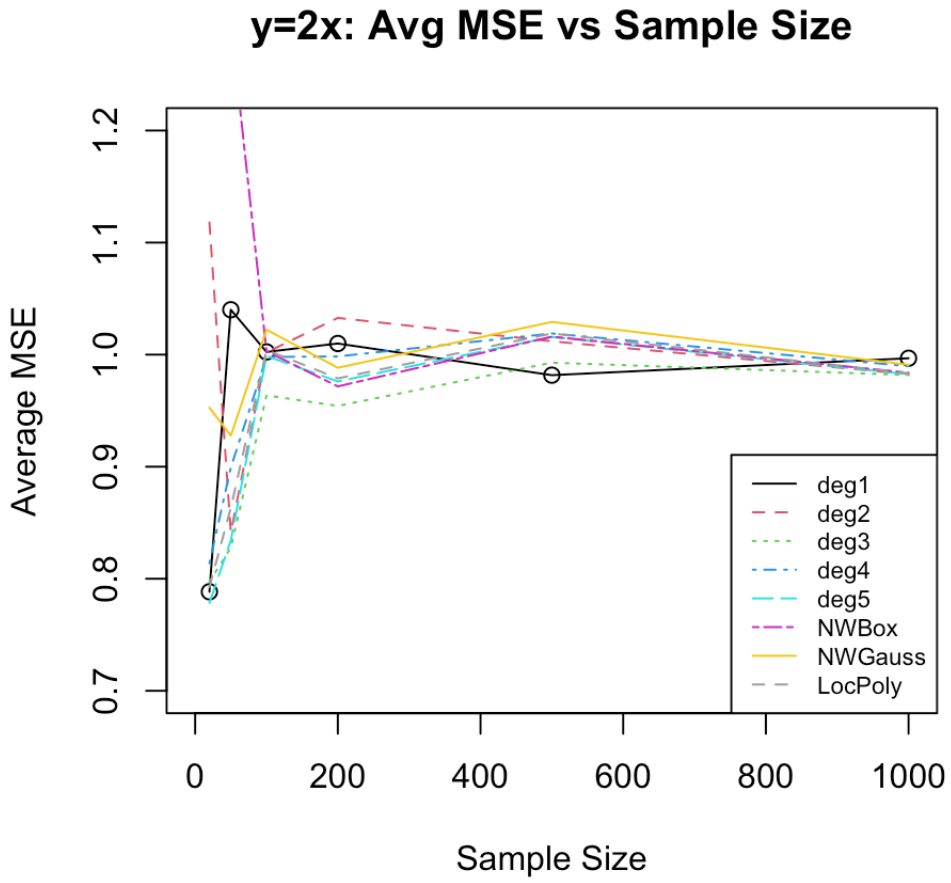
This differs from using a projection estimator without penalization parameter where the number of basis functions is cross-validated, because that is how the bias-variance tradeoff. □

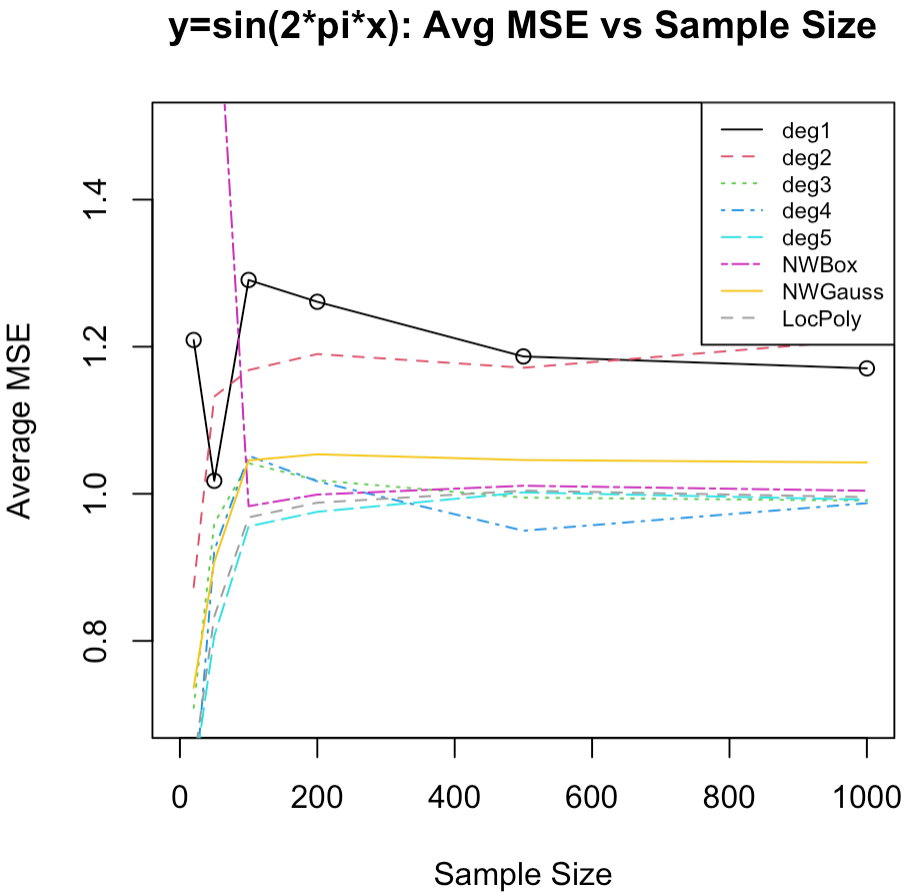
Problem 2.

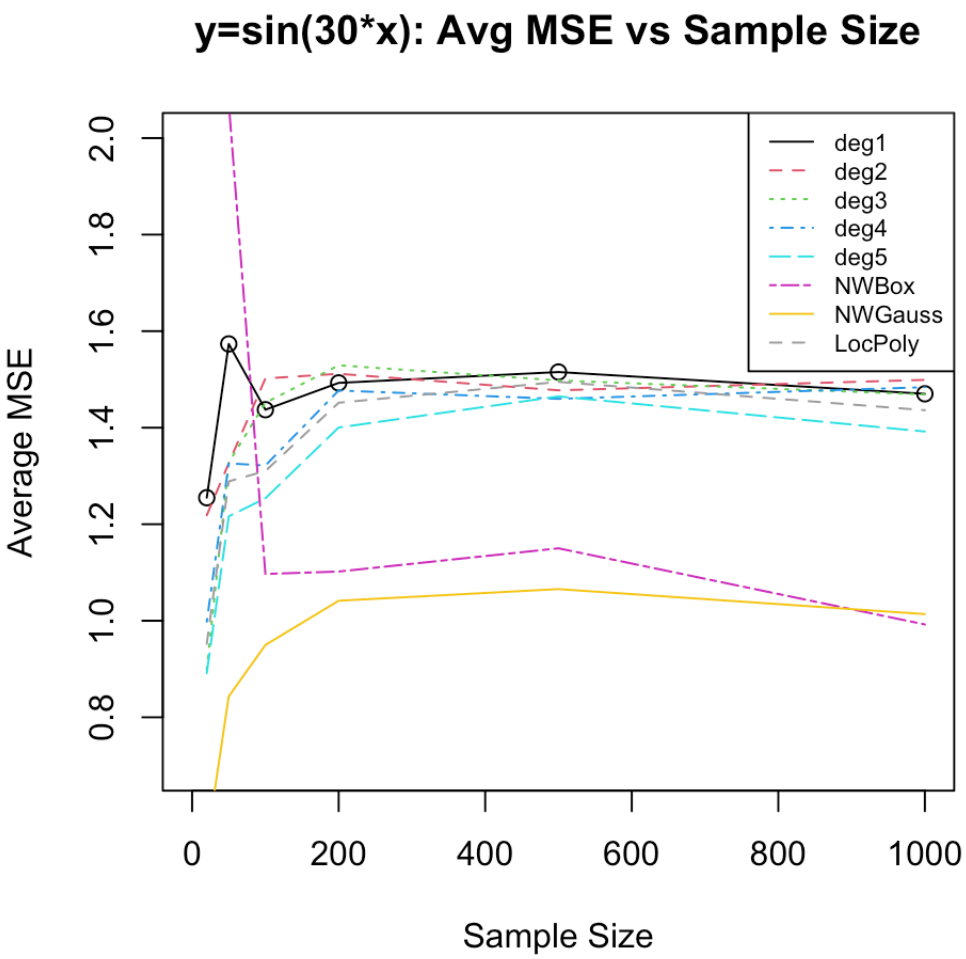
Proof. We used sample sizes of $n = 20, 50, 100, 200, 1000$ and, for each sample size, simulated 10 sets of data. Below we present the average mean square error that we found, grouped by sample size and regression method (polynomial with degrees 1-5, NW Box, NW Gaussian, and local polynomial). For NW Box and Gaussian, with the functions $y = 2x$ and $y = \sin(2\pi x)$, we used a bandwidth of 0.2 since there is low oscillation, and for $y = \sin(30x)$ we used a bandwidth of 0.05.

We point out some patterns in each of these plots. First, as the sample size becomes large, improvement in fit for all models tends to level out. In the model for $y = 2x$, as the sample sizes increases we do not see significant differences in the models' performances. This is not the case with the other functions. In $y = \sin(2\pi x)$, the degree 1 and 2 polynomial models seem to perform worse with large sample size. In $y = \sin(30x)$ the two Nadaraya-Watson models perform best.

We note that, as expected, Nadaraya-Watson with a box kernel does not perform well with small sample size because it does not have enough neighbors information to rely on. However it quickly improves with more samples. Furthermore, local polynomial regression still does not perform as well as the Nadaraya-Watson methods.




y=sin(2*pi*x): Avg MSE vs Sample Size
Average MSE
0.8
1.0
1.2
1.4
0
200
400
600
800
1000
Sample Size
deg1
deg2
deg3
deg4
deg5
NWBox
NWGauss
LocPoly


y=sin(30*x): Avg MSE vs Sample Size
Average MSE
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0
200
400
600
800
1000
Sample Size
deg1
deg2
deg3
deg4
deg5
NWBox
NWGauss
LocPoly

□

3 R Code

```
n_s = c(20,50,100,200,500,1000)
mse = array(0, dim=c(10,8,length(n_s)))
for(i in 1:length(n_s)){
 #print(i)
   for(j in 1:10){
     for(d in 1:5){
       x = sort(runif(n_s[i]))
       y = sin(30*x) + rnorm(n_s[i]) #change to compare using other truth
       functions

         reg = lm(y ~ poly(x,degree=d))
         mse[j,d,i] = (1/n_s[i])*sum((fitted(reg) - y)^2)
      }

     reg = ksmooth(x,y,bandwidth = 0.05,kernel='normal')
     mse[j,6,i] = mean((y-reg$y)^2)

     reg = numeric(n_s[i])
     box_width = .05
     for(k in 1:n_s[i]){  #create predictions vector
         reg[k] = mean(y[abs(x-x[k])<box_width])
      }
     mse[j,7,i] = mean((y - reg)^2)

     reg = loess(y ~ x)
     mse[j,8,i] = mean((y - fitted(reg))^2)
   }
 }
average_mse = array(0, dim = c(8,length(n_s)))
for(i in 1:8){
   for(j in 1:length(n_s)){
       average_mse[i,j] = mean(mse[,i,j])
     }
 }
plot(n_s, average_mse[1,],xlim=c(0,1000),ylim=c(0.7,2),xlab='Sample Size',
ylab = 'Average MSE',main='y=sin(30*x): Avg MSE vs Sample Size')
lines(n_s,average_mse[1,],lty=1)
for(i in 2:8){
   lines(n_s, average_mse[i,],col=i,lty=i)
 }
legend('topright',legend=c('deg1','deg2','deg3','deg4','deg5','NWBox',
'NWGauss','LocPoly'),col=1:8,lty=1:8,cex=0.7)
```