STAT 542 / CS 598: Homework 3

Pushpit Saxena (netid: pushpit2)

Contents

Question 1 [50 Points] A Simulation Study

- Training data $n = 30$: Generate x from $[-1, 1]$ uniformly, and then generate $y = \sin(\pi x) + \epsilon$, where ϵ's are iid standard normal.

Functions to generate training and test data:

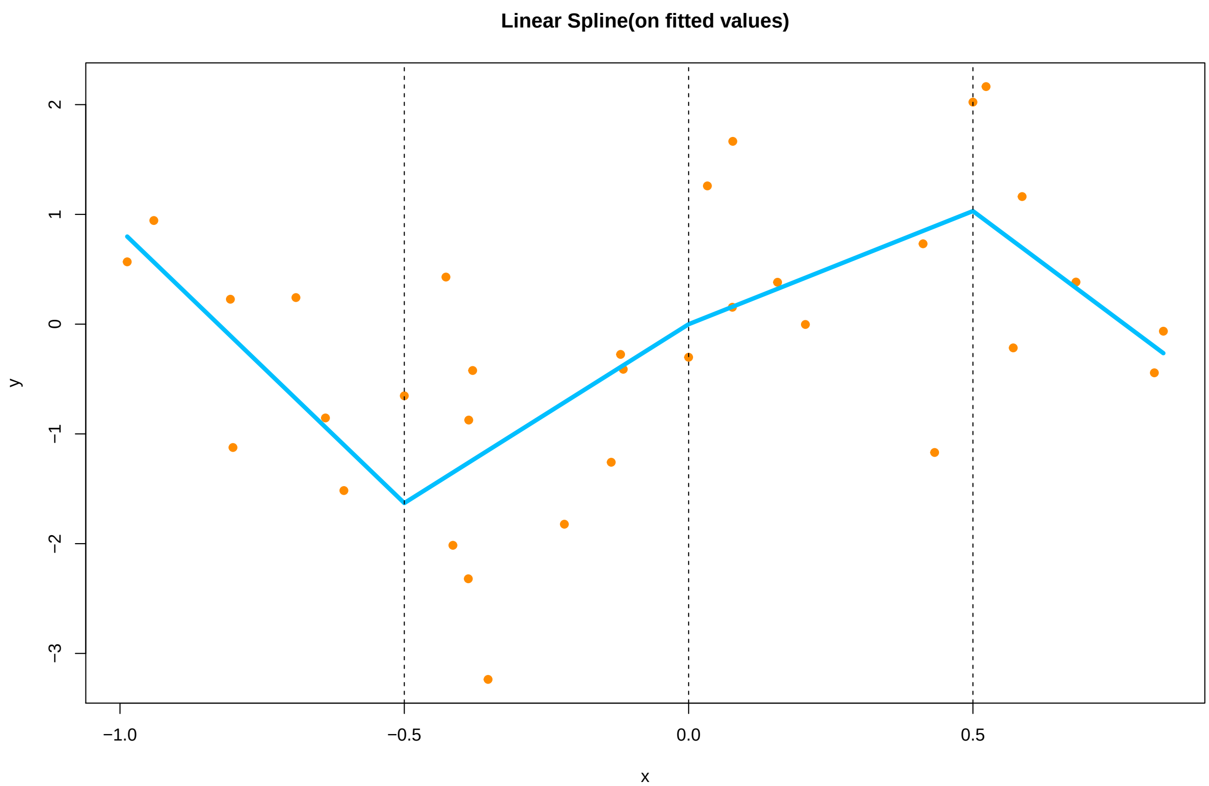
```
# True function
f <- function(x) {
  sin(pi * x)
}

# Training Data
getTrainData <- function(n) {
  x = sort(runif(n, min=-1, max=1))
  y <- f(x) + rnorm(n)
  return(list("x" = x, "y" = y))
}

# Test Data (Deterministic)
getTestData <- function(n) {
  X_test <- seq(from=-1, to=1, length.out = n)
  return(list("x" = X_test, "y" = f(X_test)))
  # Code to plot this data. I am commenting it out now in order to not clutter
  # the PDF report, but can be uncommented to see the plot.
  #plot(X_test, Y_test, type = 'l', col='green')
}
```

- Spline methods:
  - Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piecewise linear spline fitting. Choose knots at $(-0.5, 0, 0.5)$

I have followed the implementation demonstrated in the course lecture (Used the trick in basis function to make the piecewise linear spline continuous):

```
linear.spline <- function(x, x_test, y_test, myknots=c(-0.5, 0, 0.5), plot.fitted=FALSE) {
  x <- sort(append(x, myknots))
  y <- f(x) + rnorm(length(x))
  pos <- function(x) x*(x>0)
```

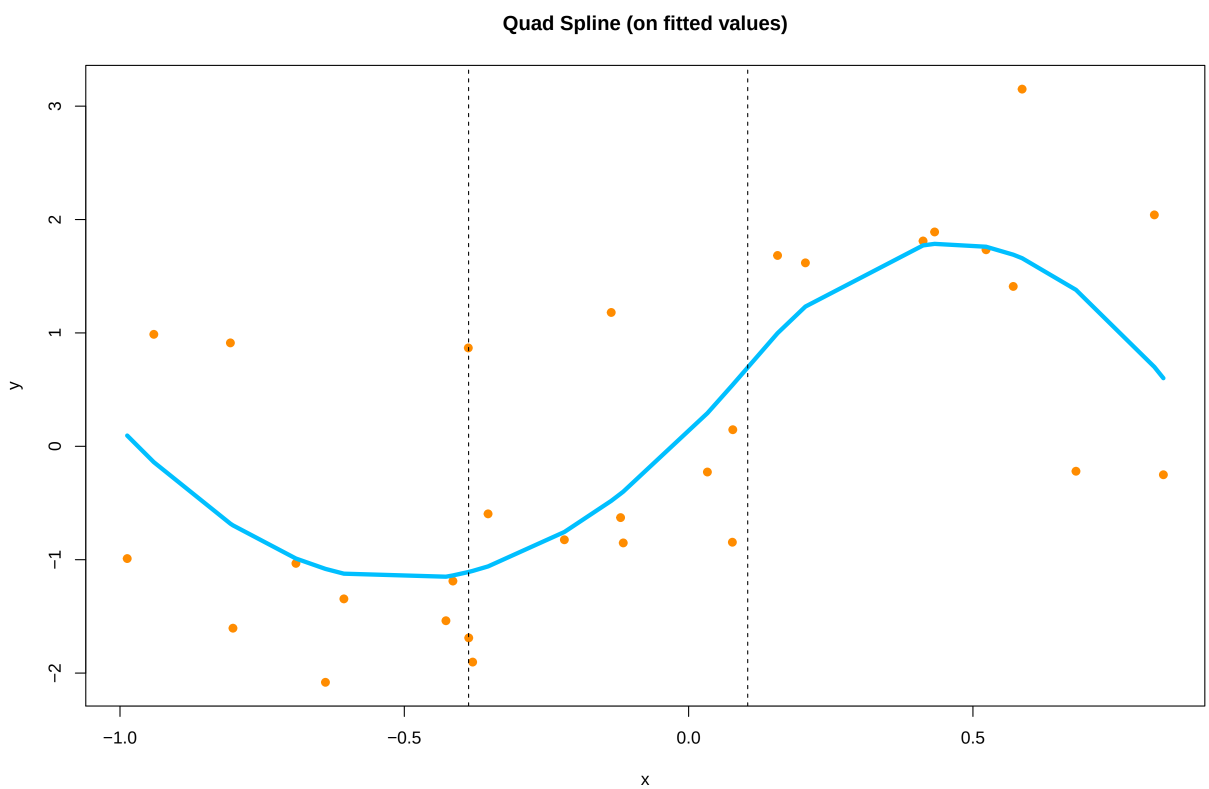
```r
  mybasis = cbind("int" = 1, "x_1" = x,
                  "x_2" = pos(x - myknots[1]),
                  "x_3" = pos(x - myknots[2]),
                  "x_4" = pos(x - myknots[3]))
  lmfit <- lm(y ~ .-1, data = data.frame(mybasis))
  mybasis_test = cbind("int" = 1, "x_1" = x_test,
                  "x_2" = pos(x_test - myknots[1]),
                  "x_3" = pos(x_test - myknots[2]),
                  "x_4" = pos(x_test - myknots[3]))
  y_pred = predict(lmfit, newdata = data.frame(mybasis_test))
  if (plot.fitted) {
    plot(x, y, pch = 19, col = "darkorange")
    lines(x, lmfit$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
    abline(v = myknots, lty = 2)
    title("Linear Spline(on fitted values)")
  }
  return (list("SSE" = sum((y_test-y_pred)^2)))
}

set.seed(1431)
train_data <- getTrainData(30)
test_data <- getTestData(1000)
error <- linear.spline(train_data$x, test_data$x, test_data$y, plot.fitted = TRUE)
```



- Use existing functions to implement a quadratic spline 2 knots. Choose your own knots.

For quadratic spline, I have used the **bs** function from the splines library. I have set the degree as 2 and defined knots are 1/3 and 2/3 quantiles of **x**

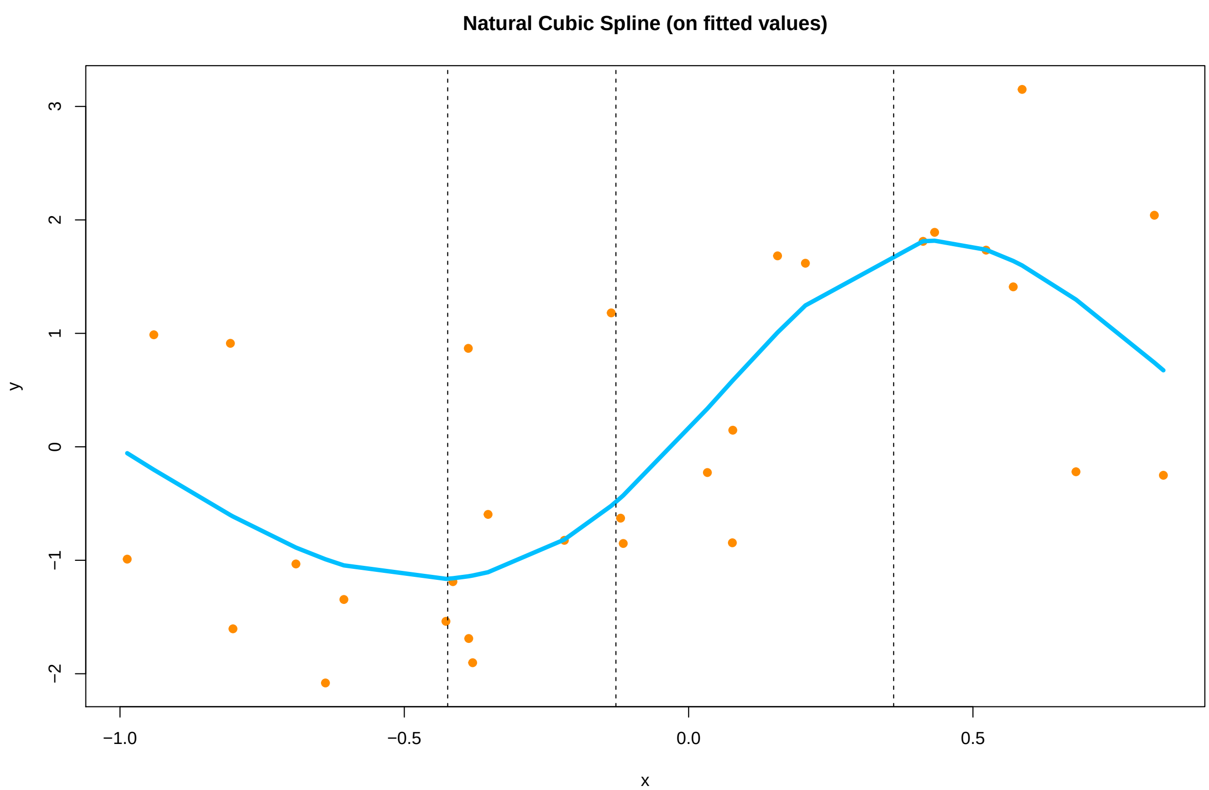
```
library(splines)
quad.spline <- function(x, y, x_test, y_test, plot.fitted=FALSE) {
  myknots <- quantile(x, probs = c(1/3, 2/3), names=FALSE)
  quad.lm <- lm(y ~ bs(x, degree = 2, knots = myknots, Boundary.knots = c(-1,1)))
  y_pred <- predict(quad.lm, data.frame(x =x_test))
  if (plot.fitted) {
    plot(x, y, pch = 19, col = "darkorange")
    lines(x, quad.lm$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
    abline(v = myknots, lty = 2)
    title("Quad Spline (on fitted values)")
  }
  return (list("SSE" = sum((y_pred - y_test)^2)))
}

error <- quad.spline(train_data$x, train_data$y, test_data$x, test_data$y, plot.fitted = TRUE)
```



- Use existing functions to implement a natural cubic spline with 3 knots. Choose your own knots.

For natural cubic spline, I have used the **ns** function from the splines library. I have defined the knots as 1/4, 1/2, 3/4 quantiles of **x**

```r
cubic.spline <- function(x, y, x_test, y_test, plot.fitted=FALSE) {
  myknots <- quantile(x, probs = c(1/4, 1/2, 3/4), names=FALSE)
  ncs.lm <- lm(y ~ ns(x, knots = myknots))
  y_pred <- predict(ncs.lm, newdata = data.frame(x = x_test))
  if (plot.fitted) {
    plot(x, y, pch = 19, col = "darkorange")
    lines(x, ncs.lm$fitted.values, lty = 1, col = "deepskyblue", lwd = 4)
    abline(v = myknots, lty = 2)
    title("Natural Cubic Spline (on fitted values)")
  }
  return(list("SSE" = sum((y_pred - y_test)^2)))
}
error <- cubic.spline(train_data$x, train_data$y, test_data$x, test_data$y, plot.fitted = TRUE)
```

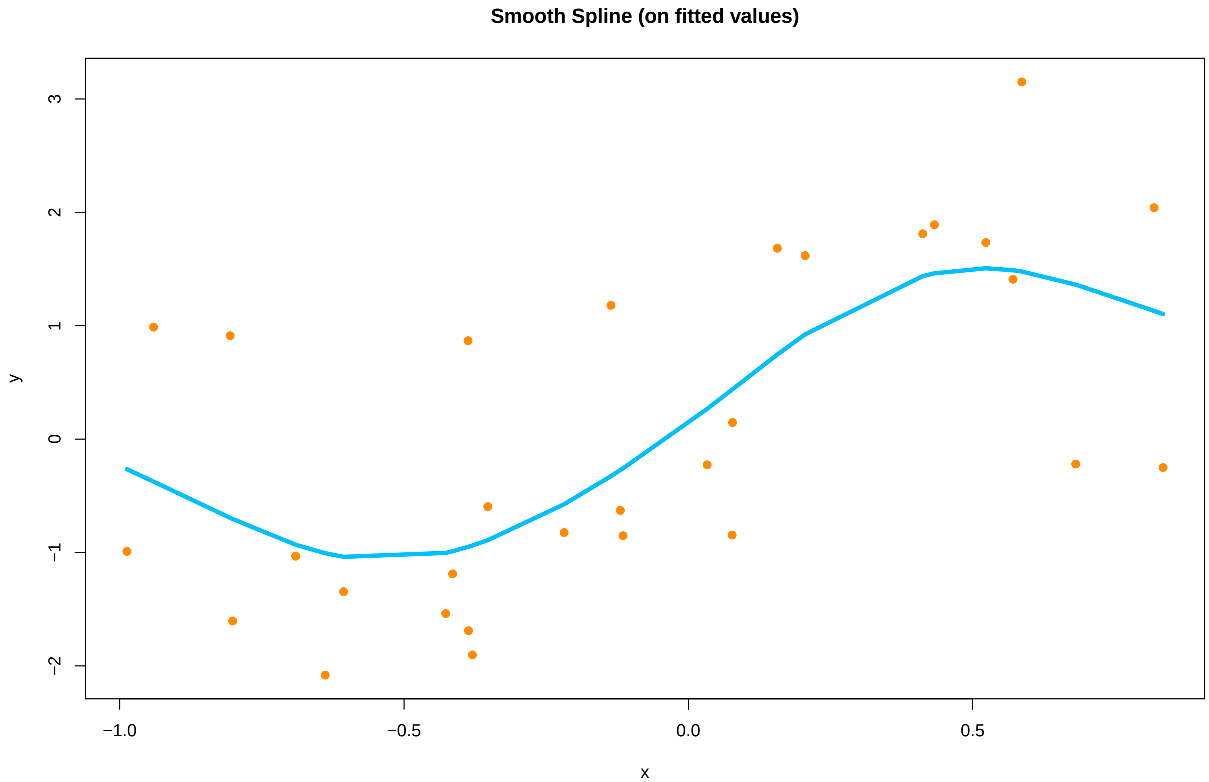


- Use existing functions to implement a smoothing spline. Use the built-in ordinary leave-one-out cross-validation to select the best tuning parameter.

I have used smooth.spline fuction from the spline library. Also, set the parameter **cv=TRUE**, so that ordinary leave-one-out cross-validation is used.

```r
my_smooth.spline <- function(x, y, x_test, y_test, cv, plot.fitted=FALSE) {
  sm <- smooth.spline(x, y, cv=TRUE)
  if (plot.fitted) {
    plot(x, y, pch = 19, col = "darkorange")
    lines(x, predict(sm, x)$y, lty = 1, col = "deepskyblue", lwd = 4)
```

```r
    title("Smooth Spline (on fitted values)")
  }
  y_pred <- predict(sm, x_test)$y
  return(list("SSE" = sum((y_pred - y_test)^2)))
}
smooth <- my_smooth.spline(train_data$x, train_data$y, test_data$x, test_data$y, plot.fitted = TRUE)
```



Training process (200 iterations, training all of the 4 models each time and recording the SSE for each run, which can be used later to show to summary statistics)

```r
set.seed(15)
test_data <- getTestData(1000)
linear_err <- numeric()
quad_err <- numeric()
cubic_err <- numeric()
smooth_err <- numeric()
for (i in 1:200) {
  train_data <- getTrainData(30)
  linear <- linear.spline(train_data$x, test_data$x, test_data$y)
  linear_err <- append(linear_err, linear$SSE)

  quad <- quad.spline(train_data$x, train_data$y, test_data$x, test_data$y)
  quad_err <- append(quad_err, quad$SSE)
```

```r
  cubic <- cubic.spline(train_data$x, train_data$y, test_data$x, test_data$y)
  cubic_err <- append(cubic_err, cubic$SSE)

  smooth <- my_smooth.spline(train_data$x, train_data$y, test_data$x, test_data$y)
  smooth_err <- append(smooth_err, smooth$SSE)
}

library(dplyr)
library(tidyr)
df = data.frame("Linear" = linear_err, "Quad" = quad_err,
                "Cubic" = cubic_err, "Smooth" = smooth_err)
data_long <- gather(df, factor_key=TRUE)
```

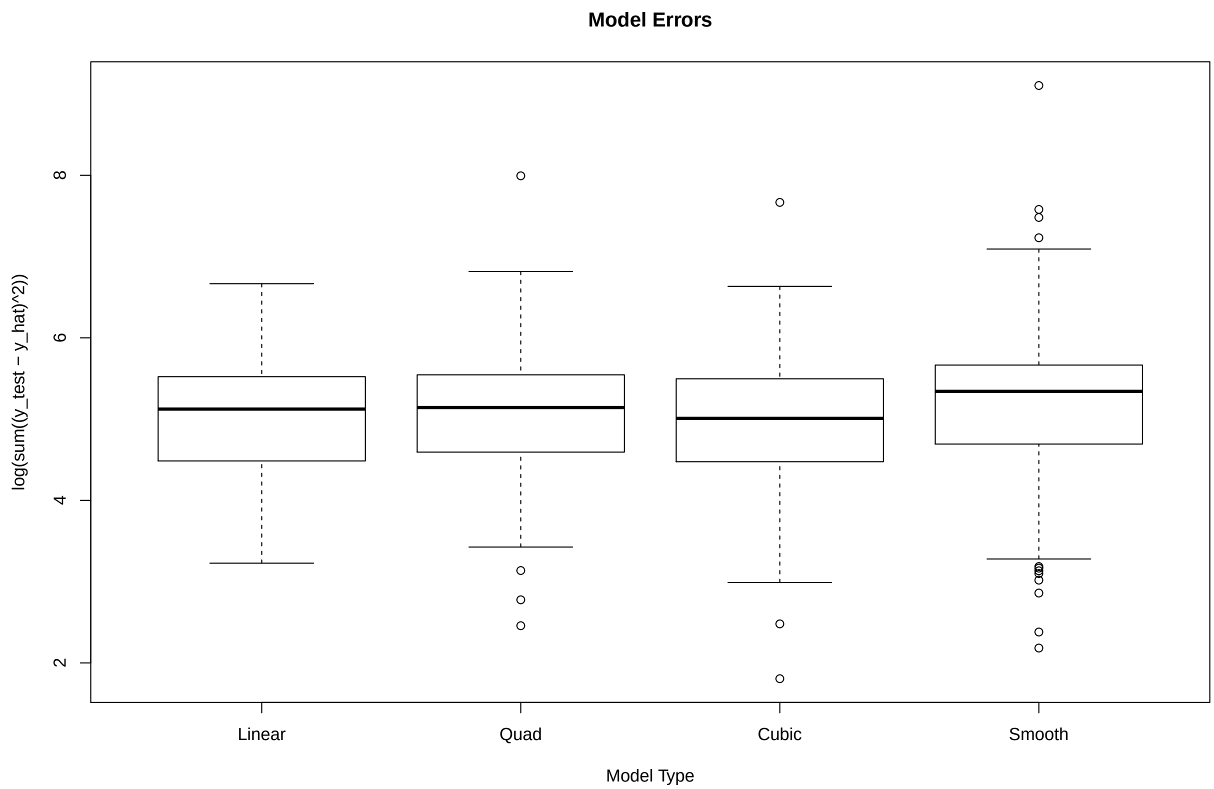
Summary statistics of the SSE (Sum of Squared Errors for all four model types):

```r
as.data.frame(data_long%>% group_by(key)%>%
  summarise(mean= mean(value), sd= sd(value), median = median(value),
            min = min(value), max=max(value)))
```

key	mean	sd	median	min	max
Linear	193.9306	137.9451	167.8905	25.210445	785.2433
Quad	221.4213	249.8019	171.1127	11.679617	2961.5558
Cubic	194.4525	191.5250	149.7278	6.085038	2135.5349
Smooth	295.3043	671.0676	208.8102	8.871778	8998.4377

BoxPlot to show errors side-by-side:

```r
boxplot(log(df), xlab="Model Type", ylab="log(sum((y_test - y_hat)^2))", main="Model Errors")
```



Comment on your findings. Which method would you prefer?

Generally, all three (Linear, Quad & Cubic) models performed similarly well but with some of the seeds (that I particularly tried) sometimes Linear comes ahead and sometimes Cubic. This entire process is very dependent on seed value selection, as well as selection of knots. I have tried varied different knots and results were sligtly different. Personally I think for the data that we have at hand, I would prefer either a Linear or Natural Cubic spline model, as it seems to fitting the data really well as well as perform better on the test data also and the training/evaluation time is also reasonable. Again, there is so much dependence on the seed/knots that its difficult to pick outright the best model with just a handful of experiments that I have done. Ideally I would like to prepare a grid of seeds as well as knots and perform the grid search, for best seed/knots for each of the model and then only I would be able to outrightly pick a best model (I hope!).

Question 2 [50 Points] Multi-dimensional Kernel and Bandwidth Selection

Data Preparation:

- Loaded the data from condvis library

```
if(!require(condvis)) install.packages("condvis", repos = "http://cran.us.r-project.org")
data(powerplant)
dim(powerplant)
```

```
## [1] 9568    5
```

- Peak into the data:

```
head(powerplant)
```

AT	V	AP	RH	PE
8.34	40.77	1010.84	90.01	480.48
23.64	58.49	1011.40	74.20	445.75
29.74	56.90	1007.15	41.91	438.76
19.07	49.69	1007.22	76.79	453.09
11.80	40.66	1017.13	97.20	464.43
13.97	39.16	1016.05	84.60	470.96

- Spliting the data into training and testing set, based on the factor (2/3) as asked in the question:

```
powerplant.mat <- model.matrix( ~ . -1, data = powerplant)

set.seed(1)
smp_size <- floor((2/3) * nrow(powerplant.mat))
train_ind <- sample(seq_len(nrow(powerplant.mat)),size = smp_size)
train_df <- powerplant.mat[train_ind,]
test_df <- powerplant.mat[-train_ind,]

X_train = train_df[, -dim(train_df)[2]]
Y_train = train_df[, dim(train_df)[2]]


X_test = test_df[, -dim(test_df)[2]]
Y_test = test_df[, dim(test_df)[2]]
```

Fitting a linear model which can be used for comparison later:

```
# Linear model which will be used for comparison later
lmfit <- lm(PE ~ ., as.data.frame(train_df))
summary(lmfit)
```

```
##
## Call:
## lm(formula = PE ~ ., data = as.data.frame(train_df))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.284  -3.144  -0.153   3.157  17.424
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 451.923114  11.902517   37.969  < 2e-16 ***
## AT           -1.962281   0.018689 -104.998  < 2e-16 ***
## V            -0.237952   0.008930  -26.645  < 2e-16 ***
## AP            0.064339   0.011546    5.572 2.62e-08 ***
## RH           -0.153891   0.005076  -30.316  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.552 on 6373 degrees of freedom
## Multiple R-squared:  0.9286, Adjusted R-squared:  0.9285
## F-statistic: 2.071e+04 on 4 and 6373 DF,  p-value: < 2.2e-16
```

```
lm_y_pred <- predict(lmfit, as.data.frame(test_df))
```

A multivariate Gaussian kernel function defines the distance between two points:

$$K_{\boldsymbol{\lambda}}(x_i, x_j) = e^{-\frac{1}{2}\sum_{k=1}^{p}((x_{ik}-x_{jk})/\lambda_k)^2}$$

- Function to calculate distance (K value) between two data points:

```
kValue <- function(Xi, Xj, lambda) {
  val <- exp(-1/2 * sum(((Xi - Xj) / lambda) ^ 2))
  return(val)
}
```

The most crucial element in kernel regression is the bandwidth λ_k. A popular choice is the Silverman formula. The bandwidth for the kth variable is given by

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

where $\widehat{\sigma}_k$ is the estimated standard deviation for variable k, p is the number of variables, and n is the sample size.

- Following is the implementation of the silverman bandwidth selection:

```
silverman.lambda <- function(X_train) {
  sd <- apply(X_train, 2, sd)
  p <- dim(X_train)[2]
  n <- dim(X_train)[1]
  return(list("lamda" = ((4 / (p + 2)) ^ (1/(p +4))) * n ^ (-1/(p+4)) * sd))
}
```

Generating silverman bandwidth selection values:

```
# Getting lambda
sl.lamda <- silverman.lambda(X_train)
sl.lamda$lamda
```

```
##       AT        V       AP       RH
## 2.371325 4.025782 1.896568 4.671344
```

Nadaraya-Watson kernel estimator implementation:

```r
# Nadaraya-Watson kernel estimator implementation
kernel.estimator <- function(xTestInstance, xtrain, ytrain, lambda) {
  kValues = apply(xtrain, 1, kValue, xTestInstance, lambda)
  return(sum(kValues * ytrain) / sum(kValues))
}
```

Fitting the model with Nadaraya-Watson kernel estimator and silverman bandwidth and comparison with linear model:

I have calculated the MSE (Mean Squared Error) and L2-norm for both kernel estimator model and linear model (fitted earlier). The Kernel estimator model is performing slightly better, when it comes to data in hand.

```r
# start.time <- Sys.time()
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, sl.lamda$lamda)
# end.time <- Sys.time()
# end.time - start.time
MSE <- mean((y_pred - Y_test) ^ 2)
MSE_LM <- mean((lm_y_pred - Y_test) ^ 2)

L2Norm <- sqrt(sum((y_pred - Y_test) ^ 2))
L2Norm_LM <- sqrt(sum((lm_y_pred - Y_test) ^ 2))

df <- data.frame("Model" = c("Kernel Estimator (SilverMan Bandwidth)", "Linear Model"),
                 "MSE" = c(MSE, MSE_LM), "L2Norm" = c(L2Norm, L2Norm_LM), stringsAsFactors = FALSE)
df
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.44354	242.5591
Linear Model	20.89601	258.1826

Bandwidth selection experiments

- Experiment 1: First experiment that I have done is to perform a simulation, setting each of the

$$\lambda_k = c(1.1, 1.2, 1.3, 1.4)$$

  and generate a grid of values. Below is one of the experiment from that simulation for which I got the best result and it is better than the one I got with using silverman lambda.

```r
lambda <- c(1.1, 1.2, 1.2, 1.3)
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, lambda)
df1 <- rbind(df, c("Kernel Estimator(Cross-validation simulation)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(Cross-validation simulation)	14.8403510038098	217.579226265178

- Experiment 2: Next experiment is done with bandwidth to be just the standard deviation * 1/(number of predictors) for each predictor. This experiment also improved the result. Although this is very similar to first experiment but here variability of each predictor is taken into account (intead of using just constant value for each predictor, as done in experiment 1).

```
p <- dim(X_train)[2]
lambda <- apply(X_train, 2, sd) * (1/ p)
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, lambda)
df1 <- rbind(df, c("Kernel Estimator(sd * 1/p as bandwidth)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(sd * 1/p as bandwidth)	16.5024462615333	229.440196073599

- Experiment 3: Selecting bandwidth as iid standard normal. This experiment performed similar to the silverman bandwidth selection.

```
lambda <- rnorm(4)
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, lambda)
df1 <- rbind(df, c("Kernel Estimator(rnorm as bandwidth)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(rnorm as bandwidth)	18.748551869154	244.556497485962

- Experiment 4: Plug-in rules (Rule of thumb) bandwidth selection. Please see here:

This bandwidth selection performed better than the silverman on the dataset we have. The use of standardized interquantile range helps in reducing the effects of potential outliers.

```
n <- dim(X_train)[1]
sd <- apply(X_train, 2, sd)
iqr <- apply(apply(X_train, 2, quantile, c(0.25, 0.75)), 2, diff)/diff(qnorm(c(0.25, 0.75)))
lambda <- 1.06 * n ^ (-1/5) * pmin(sd, iqr)
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, lambda)
df1 <- rbind(df, c("Kernel Estimator(Rule of Thumb as bandwidth)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(Rule of Thumb as bandwidth)	14.7977509972124	217.266715539007

- Experiment 5: Least-squares cross-validation (LSCV) bandwidth matrix selector for multivariate data:

I am using Hlscv function from ks library(here) for getting the bandwidth matrix selector and ran for 1st row and diagonal divided by number of predictors(4). The drawback of this selection is the time it takes to generate the bandwidth matrix for selection. This matrix provide a way to perform grid search for bandwidth params (I have conducted only 2 experiment below to keep the report clutter free as well as keep the runtime of RMD file within reasonable limits).

- Experiment 5.1: Running with first-row of bandwidth matrix (No performance improvement from Silverman, infact the perfomance decreased):

```
library(ks)
h <- Hlscv(X_train)
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, h[1,])
df1 <- rbind(df, c("Kernel Estimator(Hlscv[1] as bandwidth)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(Hlscv[1] as bandwidth)	26.2036548095301	289.118762522257

- Experiment 5.2: Running with diagonal of bandwidth matrix divided by the number of predictors (Performance improved!):

```
hh <- diag(h) / dim(X_train)[2]
y_pred <- apply(X_test, 1, kernel.estimator, X_train, Y_train, hh)
df1 <- rbind(df, c("Kernel Estimator(diag(Hlscv)/p as bandwidth)",
                   mean((y_pred - Y_test) ^ 2), sqrt(sum((y_pred - Y_test) ^ 2))))
df1
```

Model	MSE	L2Norm
Kernel Estimator (SilverMan Bandwidth)	18.4435426539083	242.559067169148
Linear Model	20.8960122546489	258.182646768388
Kernel Estimator(diag(Hlscv)/p as bandwidth)	14.854759346345	217.684823345222

Conclusion on bandwidth selection:

The best result (not by very far) is from the bandwidth selection in **Experiment 4: Rule of Thumb** as per the experiments that I have ran. There are variety of other methods for bandwidth selection and due to time (report space) constraints, I haven't experimented with all of them. Generally there is no exact and universal answer to bandwidth selection. The best way is to try different selectors and run experiments to determine the best selector based on kernel estimation results. Each of the methods above have some deficiencies. The conclusion (see here) summarizes the problem of bandwidth selection very aptly.