Statistical Machine Learning - Coursework I

Name: Harleen Gulati , Student Number: 2101550

03/02/2024

Problem I

Problem I (i)

Applying the OLS estimate for β on the augmented data-set given in the question yields:

$$\hat{\beta}^{OLS} = (\mathbf{\tilde{X}^T\tilde{X}})^{-1}\mathbf{\tilde{X}^T\tilde{y}} \ (*)$$

Now:

$$\begin{aligned}\mathbf{\tilde{X}^T\tilde{X}} &= \begin{bmatrix}\mathbf{X^T} & \sqrt{\lambda}\mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{X} \\ \sqrt{\lambda}\mathbf{I}\end{bmatrix} \\ &= \mathbf{X^TX} + \lambda\mathbf{I} \ (**)\end{aligned}$$

and

$$\begin{aligned}\widetilde{\mathbf{X}^T}\mathbf{\tilde{y}} &= \begin{bmatrix}\mathbf{X^T} & \sqrt{\lambda}\mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{y} \\ \mathbf{0}\end{bmatrix} \\ &= \mathbf{X^Ty} \ (***)\end{aligned}$$

Substituting

$$(**)$$

and

$$(***)$$

into $(*)$ gives

$$\begin{aligned}\hat{\beta}^{OLS} &= (\mathbf{X^TX} + \lambda\mathbf{I})^{-1}\mathbf{X^Ty} \\ &= \hat{\beta}^{\mathbf{r}}_{\lambda}\end{aligned}$$

as required.

Problem I (ii) (a)

For any value of α and $\lambda = 0$ we have:

$$\hat{\beta}_{(0,\alpha)} = argmin_{\beta}[||\mathbf{y} - \mathbf{X}\beta||^2] = \hat{\beta}^{OLS}$$

which is precisely the OLS estimator.

For any value of $\lambda \geq 0$ and $\alpha = 1$ we have

$$\hat{\beta}_{(\lambda,1)} = argmin_\beta[||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda||\beta||_2^2] = \hat{\beta}_\lambda^r$$

which is precisely the ridge regression estimator.

For any value of $\lambda \geq 0$ and $\alpha = 0$ we have

$$\hat{\beta}_{(\lambda,0)} = argmin_\beta[||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda||\beta||_1] = \hat{\beta}_\lambda^1$$

which is precisely the LASSO estimator.

In summary as $\lambda \rightarrow 0$, Elastic Net becomes similar to OLS, introducing less penalty on the β terms.

As $\alpha \rightarrow 1$, Elastic Net becomes similar to ridge regression (shrinking coefficients to 0 but not to absolute 0).

As $\alpha \rightarrow 0$, Elastic Net becomes similar to LASSO (coefficients may be shrunk to exactly 0).

Problem I (ii) (b)

Idea: To obtain the elastic net estimate $\hat{\beta}_{\lambda,\alpha}^e$ via the LASSO on an augmented data set, we need to find an augmentation such that $\hat{\beta}_{\lambda,\alpha}^e$ loses the $\lambda\alpha||\beta||_2^2$.

Perhaps having $\tilde{\mathbf{X}}$ be an nx2p matrix where each row i contains $x_{i,1}, \ldots, x_{i,p}$ in addition to p extra rows, where each of these extra rows contain $\lambda\alpha$.

Problem I (iii) (a)

In the scenario where q = 2 and we recover the ridge regression, there are no 0 valued entries in $\tilde{\beta}_{(\lambda,q)}$ (ridge regression shrinks the parameters towards 0, but doesn't set them exactly to 0).

In the scenario where q = 1 and we recover the LASSO, there may be some 0 valued entries in $\tilde{\beta}_{(\lambda,q)}$ (LASSO has the ability to set some parameters exactly equal to 0).

For q = 2, as lambda increases more of the parameters get closer to 0 and for q = 1 as lambda increases more of the entries in $\tilde{\beta}_{(\lambda,q)}$ get set to 0.

For q = 0, our problem becomes

$$\tilde{\beta}_{(\lambda,q)} = argmin_\beta[||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda \sum_{i=1}^{p} |\beta|^0$$
$$= argmin_\beta[||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda p$$

Now as $\lambda > 0$ and p > 0 => $\lambda p > 0$. Thus $\lambda p > 0$ is a fixed constant and so our problem focuses on $argmin_\beta[||\mathbf{y} - \mathbf{X}\beta||^2$ which yields the OLS estimator which could yield 0 valued entries in $\tilde{\beta}_{(\lambda,q)}$ dependent upon the data.

For q < 1, $\sum_{i=1}^{p} |\beta_i|^q$ is concave.Thus, small values of β_i are closer to 0, and eventually β_i flattens, implying that the rate of the penalty decreases as β_i increases. This thus implies that the coefficients for q < 1 do not shrink aggressively towards 0, however slowly decrease towards 0 and some may even be 0.

Problem I (iii) (b)

We are penalizing differences between consecutive elements of β - this penalization will result in a solution where adjacent elements of β are closer in value.

The solution will therefore be smoother, because if adjacent elements in β are closer together, then a small change in the predictor variable will lead to a small change in the response.

Problem II

Loading the data

```
load(file='regressioncoursework.Rdata')
n = nrow(data)
x = data$x
y = data$y
observations = data # rename data to observations
```

Problem 2 (i) (a)

The linear model we obtain is

$$y = 3.863552 + 1.977070x$$

with coefficients being 3.863552 (intercept) and 1.977070 (x value coefficient) respectively

```
model = lm(y ~ x, data=observations)
coefs = model$coefficients
print(coefs)
```

```
## (Intercept)           x
##    3.863552    1.977070
```

Problem 2 (i) (b)

The OLS estimator gives us coefficients 3.86355 (intercept) and 1.977070 (x value coefficient) respectively. Thus lm and the OLS estimator yield the same coefficients implying that lm finds a linear fit by minimizing the mean squared error (i.e., by using the same method the OLS estimator uses to find a linear fit).

```
interceptcol = rep(1, n) # creates a list of 1s n times
X = cbind(interceptcol, x) # creates the design matrix where the first column is a li
st of n 1s (for the intercept) and the second column is the x values
X_T = t(X) # transpose of X
beta_hat_OLS = solve(X_T %*% X) %*% X_T %*% y
print(beta_hat_OLS)
```

```
##                  [,1]
## interceptcol 3.863552
## x            1.977070
```

Problem 2 (i) (c)

The mean squared error of the linear model using the fitted values

$$\tilde{y}_i = \mathbf{X}\hat{\beta}^{\mathbf{(OLS)}}$$

is 0.4803235

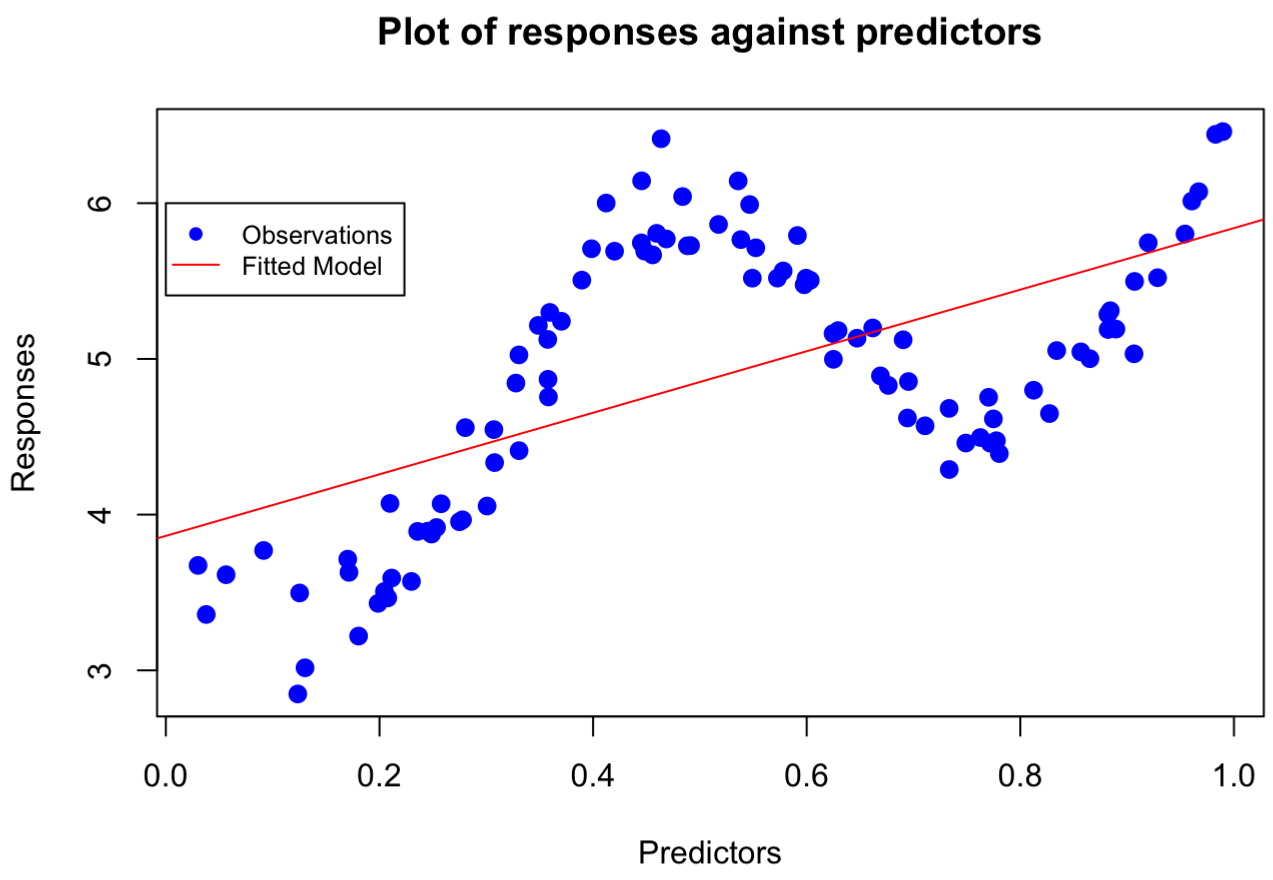
```
fittedvals = X %*% beta_hat_OLS
residuals = y - fittedvals
meansquarederror = 1/n * (t(residuals) %*% residuals)
print(meansquarederror)
```

```
##           [,1]
## [1,] 0.4803235
```

Problem 2 (i) (d)

We plot the observations which are shown as blue dots and the fitted model shown as the red line. The simple linear model under fits the data as it passes through few observations and does not capture the pattern of the data, thus we see the simple linear model may not be appropriate.

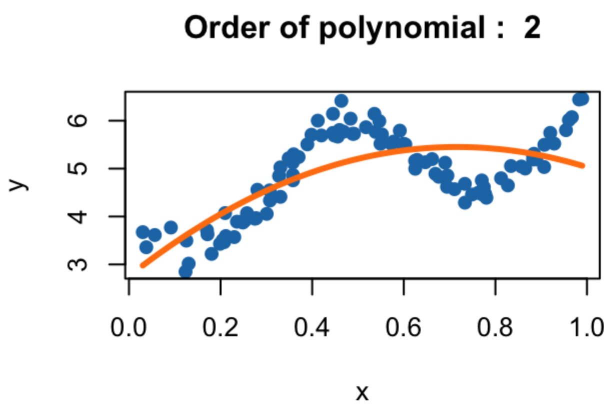
```
plot(observations , pch = 16, cex = 1.3, col = "blue", main = "Plot of responses agai
nst predictors", xlab = "Predictors", ylab = "Responses")
abline(model, col="red")
legend(0, 6, legend=c("Observations", "Fitted Model"),
       col=c("blue", "red"), pch=c(19, NA), lty=0:1, cex=0.8)
```

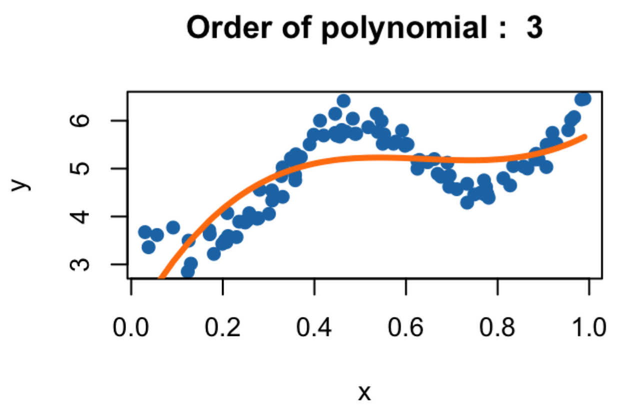


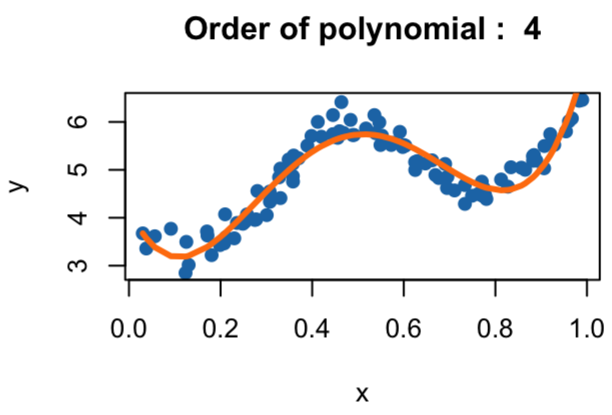
Problem 2 (ii) (a)

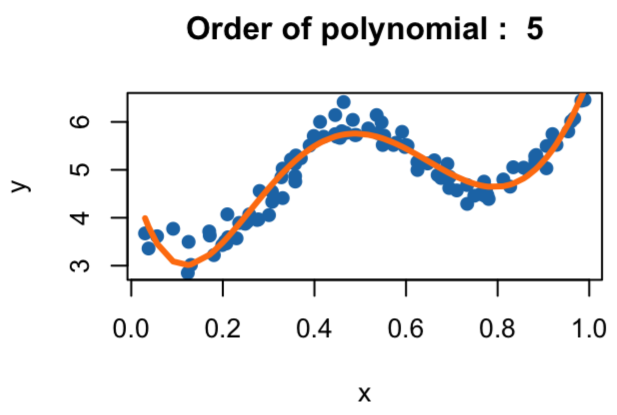
We observe how for a 2nd and 3rd order degree fitted polynomial, the fitted model underfits the data however for an order 4 and 5 degree polynomial the fitted model captures the data well thus we see how as k increases, the model becomes more accurate in fitting the data.

```
par(mfrow=c(2,2))
for (k in 2:5) {
  kth_order_model = lm(y ~ poly(x, k), observations)
  plot(data, pch = 19, col = "#1f77b4", main = paste("Order of polynomial : " , k))
  lines(x, kth_order_model$fitted.values, lty = 1, col = "#ff7f0e", lwd = 3)
}
```









Problem 2 (ii) (b)

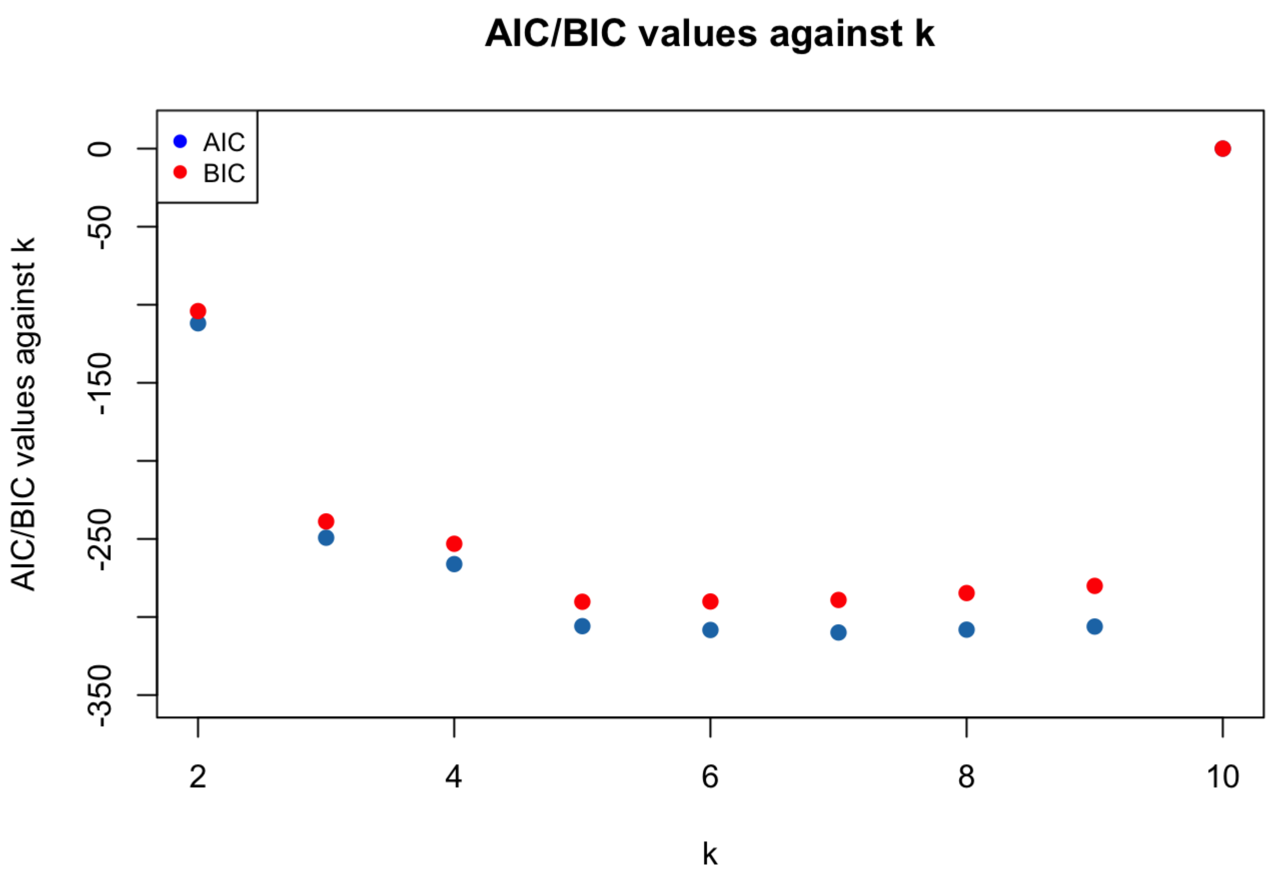
AIC and BIC are criteria for indirectly estimating the test error by accounting for the bias due to over fitting which may arise from having a polynomial with too large a complexity. Thus, we prefer a complexity which yield a smaller AIC and BIC.

We observe from the graph that the AIC is slightly smaller than BIC for each value of k, which is due to the penalization - $log(10) \approx 2.30$ thus $2k < log(n) * k$ and hence AIC is slightly smaller than BIC for each value of k.

We further observe how increasing the value of k decreases the AIC and BIC rapidly from around k = 2 to k = 5, where the AIC and BIC then stays roughly similar until both the AIC and BIC increase rapidly to 0 for k = 10.

This thus suggests that k = 5 is the optimal choice for the degree of polynomial as this is where the AIC and BIC are both minimal; increasing k beyond this point makes little difference to the AIC and BIC.

```
AIC = c(rep(0,9))
BIC = c(rep(0,9))
k_vals = c(2,3,4,5,6,7,8,9,10)
for (k in k_vals) {
  kth_order_model = lm(y ~ poly(x, k), observations)
  residuals = kth_order_model$residuals
  rss = t(residuals) %*% residuals
  AIC[k-2] = (n*log(rss/n) + 2*k)
  BIC[k-2] = (n*log(rss/n) + k*log(n))
}
plot(k_vals, AIC,  pch = 19, col = "#1f77b4", main="AIC/BIC values against k", xlab =
"k", ylab = "AIC/BIC values against k", ylim=c(-350,10), xlim=c(2,10))
points(k_vals, BIC, pch=19, col="red")
legend("topleft", legend=c("AIC", "BIC"),
       col=c("blue", "red"), pch=c(19, 19), cex=0.8)
```



Problem 2 (ii) (c)

```
split_data = sample(seq_len(n), size = 70)
train = observations[split_data,]
test = observations[-split_data,]
```

Problem 2 (ii) (d)

From the table we observe how as k increases, the training mean squared error (MSE) decreases. The greater the complexity of the polynomial, the more flexible the polynomial is and thus the better the polynomial can fit the training data.

We also observe the testing MSE decreases from k = 2 to k = 5 and then begins to increase from this point onwards, suggesting for k > 5 the model overfits to the training data capturing the noise in the training data rather than a general pattern.

For k = 10, we observe the training MSE to be 0 implying the model fits the data perfectly, which suggests the model overfits the data (fits every minor variation and noise). Thus, we can see that k = 5 is an optimal choice as this minimizes the testing MSE, and any value beyond this results in over fitting to the training data.

```
k_vals = c(2,3,4,5,6,7,8,9,10)
training_mse = c(rep(0,9))
testing_mse = c(rep(0,9))
for (k in k_vals) {
  kth_order_model = lm(y ~ poly(x, k), train)
  training_residuals = resid(kth_order_model, type = "response")
  training_mse[k-2] = mean(training_residuals^2)
  predict_test = predict(kth_order_model, test)
  testing_residuals = test$y - predict_test
  testing_mse[k-2] = mean(testing_residuals^2)
}
df <- data.frame(k_vals, training_mse, testing_mse)
print(df)
```

```
##   k_vals training_mse testing_mse
## 1      2   0.29295276  0.35623029
## 2      3   0.07590337  0.08562776
## 3      4   0.06442857  0.06744705
## 4      5   0.04020591  0.04840359
## 5      6   0.03801025  0.04776485
## 6      7   0.03675172  0.04582269
## 7      8   0.03651035  0.04644528
## 8      9   0.03619698  0.04767983
## 9     10   0.00000000  0.00000000
```

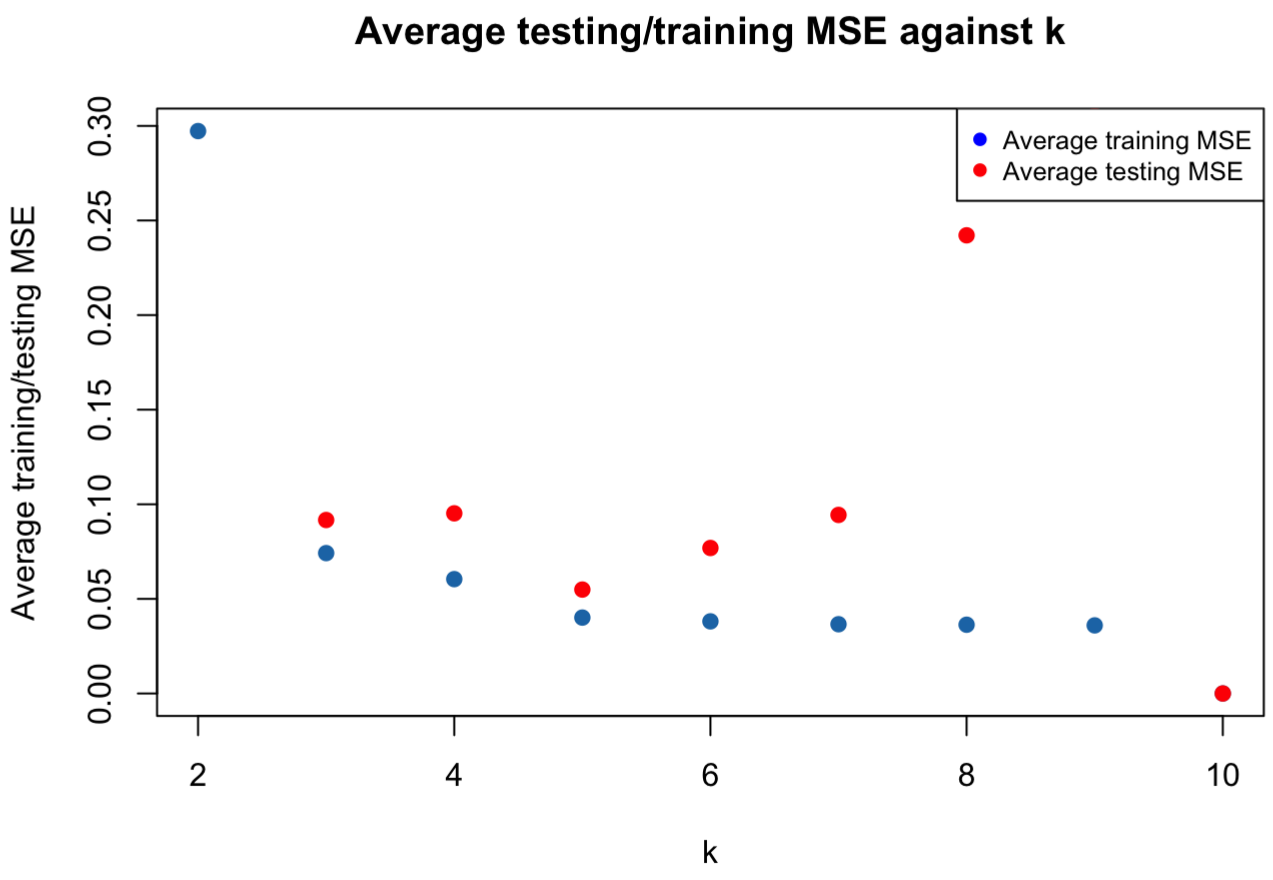
Problem 2 (ii) (e)

Our findings show the testing error to be always greater than or equal to the training error. This is due to the fact the data is trained on the training set, and thus on average would fit the training set better than the testing set.

We also observe how as the value of k increases, there is greater variability in the average training and testing MSE - the average testing MSE increases as k increases whereas the average training MSE decreases as k increases. This is due to the over fitting which results in a larger degree polynomial (the greater the degree of the polynomial, the more flexible the polynomial is which risks over fitting as minor variations are more likely to be captured). Thus, this over fitting causes the average training MSE to decrease as the polynomial better fits to the training data, however the average testing MSE increases since the model captures less of the pattern of the data and focuses more on minor variation in the data.

For k = 10, we observe the average training MSE to be 0 implying the model fits the data perfectly, which suggests the model overfits the data (fits every minor variation and noise).

```
sum_train = c(rep(0,9))
sum_test = c(rep(0,9))
k_vals = c(2,3,4,5,6,7,8,9,10)
for (rep in 1:1000) {
  split_data = sample(seq_len(n), size = 70)
  train = observations[split_data,]
  test = observations[-split_data,]
  for (k in k_vals) {
    kth_order_model = lm(y ~ poly(x, k), train)
    training_residuals = resid(kth_order_model, type = "response")
    training_mse = mean(training_residuals^2)
    sum_train[k-2] = training_mse + sum_train[k-2]
    predict_test = predict(kth_order_model, test)
    testing_residuals = test$y - predict_test
    testing_mse = mean(testing_residuals^2)
    sum_test[k-2] = testing_mse + sum_test[k-2]
  }
}
average_train_MSE = sum_train/1000
average_test_MSE = sum_test/1000
plot(k_vals, average_train_MSE, pch = 19, col = "#1f77b4", main="Average testing/trai
ning MSE against k", xlab = "k", ylab = "Average training/testing MSE")
points(k_vals, average_test_MSE, pch=19, col="red")
legend("topright", legend=c("Average training MSE", "Average testing MSE"),
       col=c("blue", "red"), pch=c(19, 19), cex=0.8)
```

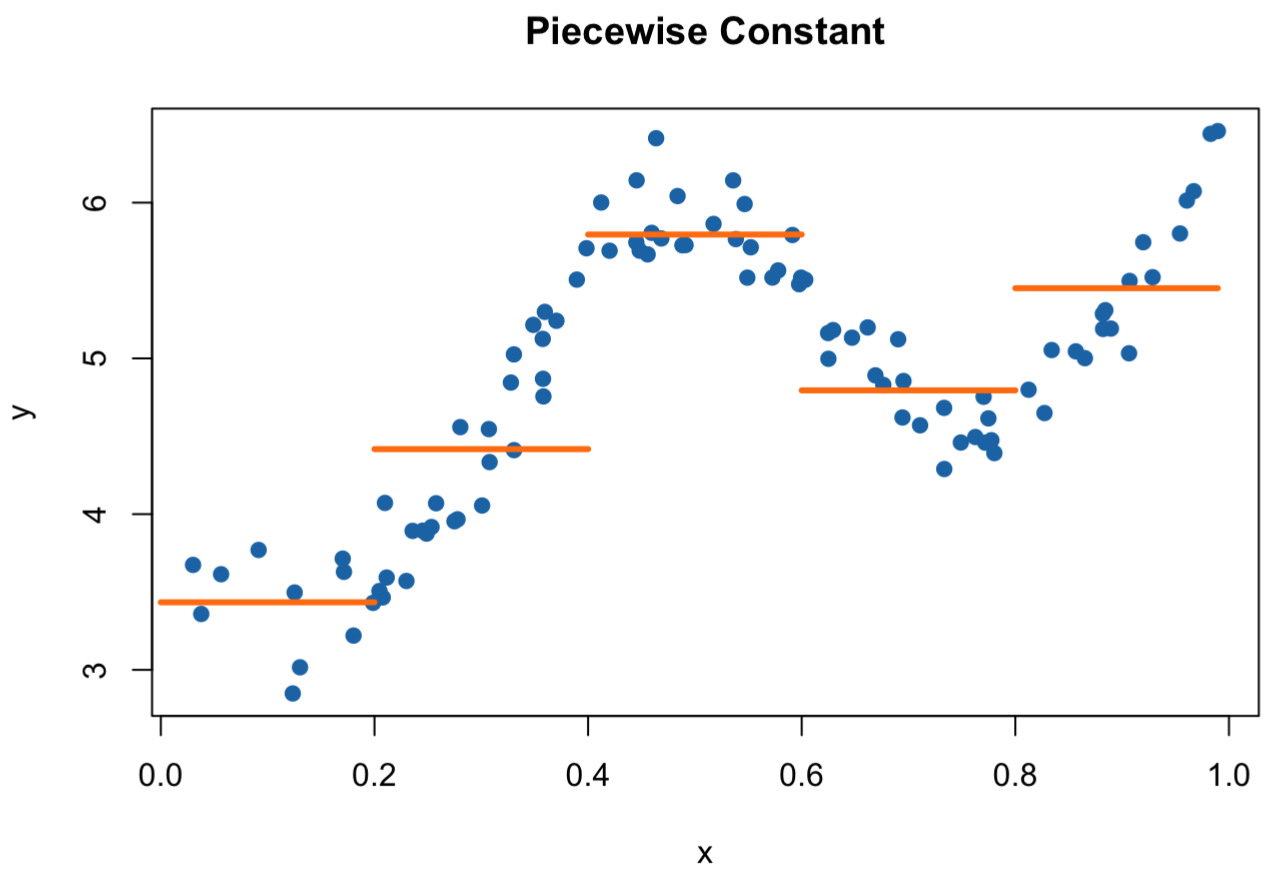


Problem 2 (iii) (a & b)

There are 5 basis functions thus $M = 5$ and these functions are $h_1(X) = \mathbb{1}(X < 0.2)$, $h_2(X) = \mathbb{1}(0.2 \leq X < 0.4)$, $h_3(X) = \mathbb{1}(0.4 \leq X < 0.6)$, $h_4(X) = \mathbb{1}(0.6 \leq X < 0.8)$, $h_5(X) = \mathbb{1}(0.8 \leq X)$

The piece wise constant polynomials are at the average of the samples in each region - however there is no smoothness and discontinuity occurs at the limits of each boundary.

```
knots = c(0.2, 0.4, 0.6, 0.8)
bounds = c(0, knots, max(observations$x))
basis = c((observations$x < knots[1]))
for (i in 1:(length(knots) - 1)) {
  basis = cbind(basis,(observations$x >= knots[i]) * (observations$x < knots[i +1]))
}
basis = cbind(basis, observations$x >= knots[length(knots)])
piecewise_constant <- lm(observations$y ~ . - 1, data = data.frame(basis))
plot(observations, pch = 19, col = "#1f77b4", main = "Piecewise Constant")
for (k in 1:length(bounds)) {
  points(c(bounds[k], bounds[k + 1]), rep(piecewise_constant$coefficients[k], 2), typ
e = "l", lty = 1, col = "#ff7f0e", lwd = 3)
}
```



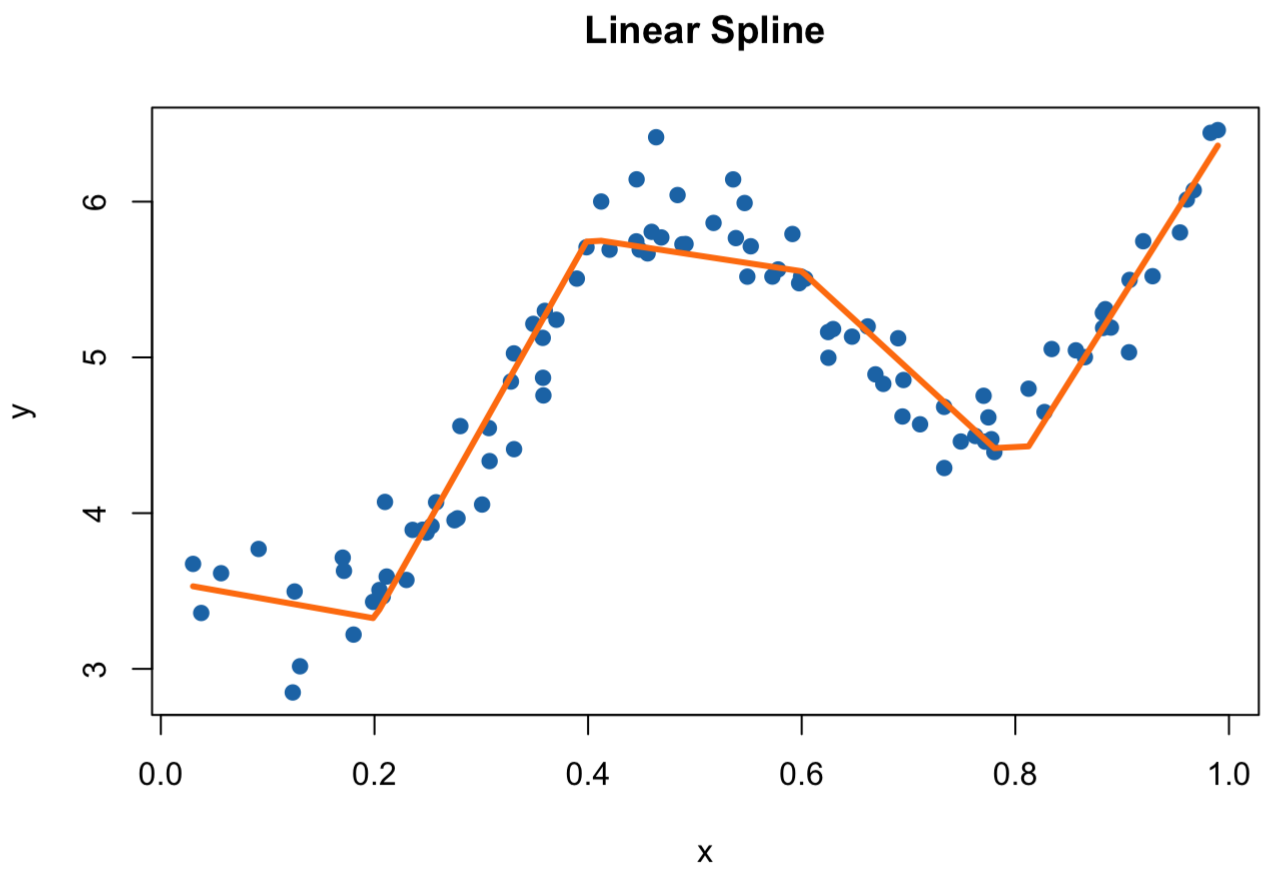
Problem 2 (iii) (c)

The basis functions are as follows:

$h_1(X) = \mathbb{1}(X < 0.2)$ $h_2(X) = \mathbb{1}(0.2 \leq X < 0.4)$ $h_3(X) = \mathbb{1}(0.4 \leq X < 0.6)$ $h_4(X) = \mathbb{1}(0.6 \leq X < 0.8)$ $h_5(X) = \mathbb{1}(0.8 \leq X)$ $h_6(X) = \mathbb{1}(X < 0.2)X$ $h_7(X) = \mathbb{1}(0.2 \leq X < 0.4)X$ $h_8(X) = \mathbb{1}(0.4 \leq X < 0.6)X$ $h_9(X) = \mathbb{1}(0.6 \leq X < 0.8)X$ $h_{10}(X) = \mathbb{1}(0.8 \leq X)X$

Thus there are 10 basis functions and we observe the model is now continuous.

```
library(splines)
knots = c(0.2, 0.4, 0.6, 0.8)
linear_spline <- lm(observations$y ~ bs(observations$x, degree = 1, knots = knots), d
ata = observations)
plot(observations, pch = 19, col = "#1f77b4")
lines(observations$x, linear_spline$fitted.values, lty = 1, col = "#ff7f0e", lwd = 3)
title("Linear Spline")
```

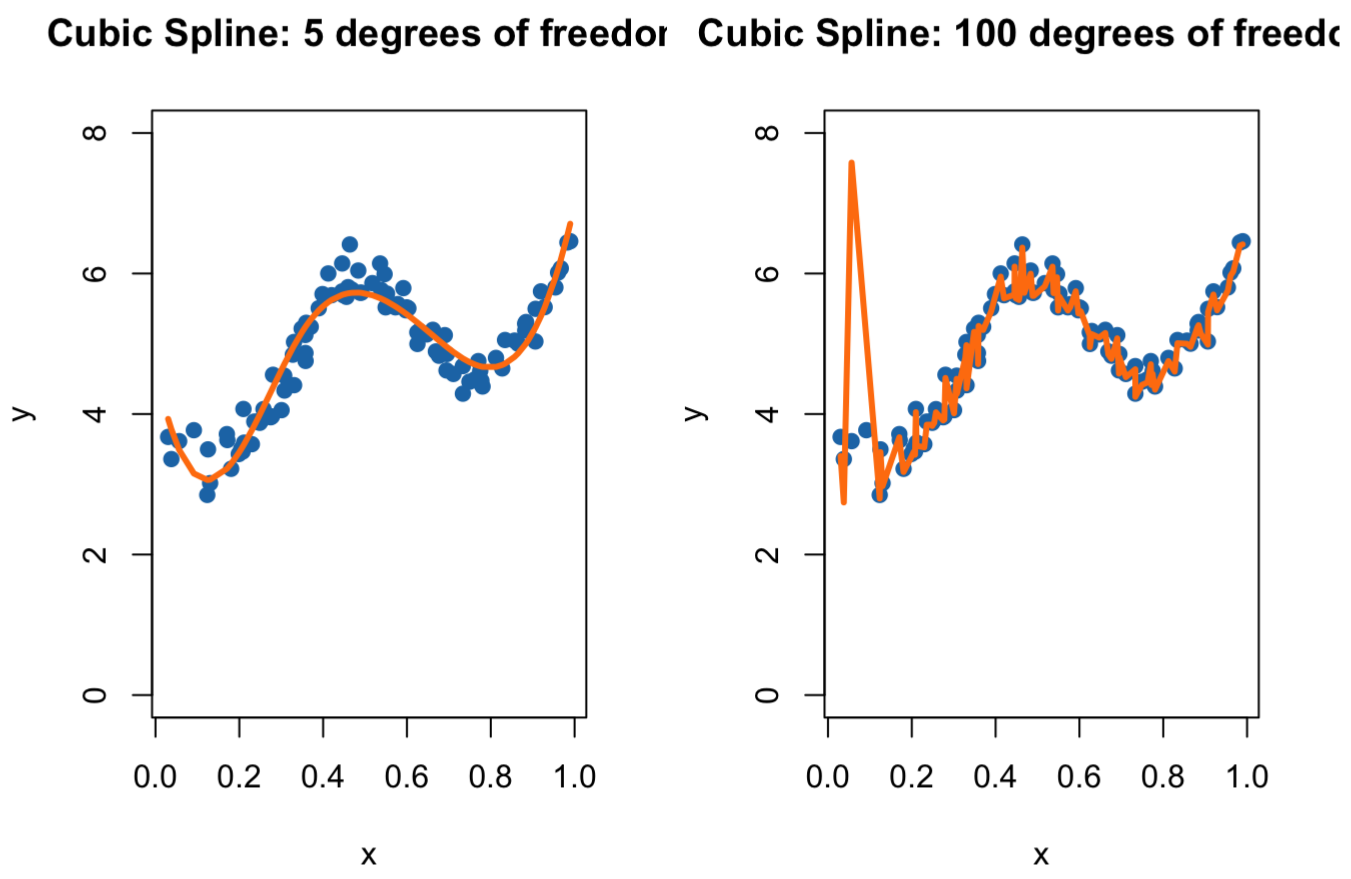


Problem 2 (iii) (d)

We observe how a cubic smoothing spline with 5 degrees of freedom captures the general pattern of the data well however with 100 degrees of freedom overfits to the data, capturing every minor variation and noise. We also observe how the cubic smoothing spline with 5 degrees of freedom is a lot more smoother, whereas the cubic smoothing spline with 100 degrees of freedom has more spikes and and is more jagged.

This is because as the degrees of freedom increase, the number of basis functions and thus coefficients available to model the data increase thus increasing the flexibility of the model which risks overfitting to minor variations/noise. With an increase in the degrees of freedom, since we are more likely to model minor variations/noise, we also observe how the smoothness of the model is affected and hence decreases.

```r
library(splines)
options(warn=-1)
cubic_spline_fivedof <- lm(observations$y ~ bs(observations$x, degree = 3,
df = 5), data = observations)
cubic_spline_100dof <- lm(observations$y ~ bs(observations$x, degree = 3,
df = 100), data = observations)
par(mfrow = c(1, 2))
plot(observations, pch = 19, col = "#1f77b4", ylim=c(0,8))
lines(observations$x, predict(cubic_spline_fivedof, data.frame(observations$x)), lty
= 1, col = "#ff7f0e", lwd = 3)
title("Cubic Spline: 5 degrees of freedom")
plot(observations, pch = 19, col = "#1f77b4" ,ylim=c(0,8))
lines(observations$x, predict(cubic_spline_100dof, data.frame(observations$x)), lty =
1, col = "#ff7f0e", lwd = 3)
title("Cubic Spline: 100 degrees of freedom")
```



Problem 2 (iii) (e)

The code below uses leave one out cross-validation to determine a suitable integer value for the degrees of freedom (df).

In summary the code does as follows:

- we have a list of degrees of freedom we'd like to test (between 3 to 50 degrees of freedom)
- we aim to choose the degree of freedom which gives the smallest MSE w.r.t leave-one out cross validation (this is because we also want to penalize too many degrees of freedom as this leads to overfitting to the training data)
- for each degree of freedom we do as follows:

- we perform leave-one out cross validation (i.e., we train a cubic spline with the specified number of degrees of freedom on some training set 100 times, where each time the training set excludes a given row. This excluded row is then used as a validation set, to get a prediction $\hat{y}$ for the observation x in that row. After we have done this 100 times, we find the residual sum of squares from the true values of y to the predicted values of y and thus use this to calculate the MSE for the specified number of degrees of freedom).
- This process is repeated for all the possible degrees of freedom we specify, and we then choose the degree of freedom which gave the smallest MSE.

```
library(splines)
options(warn=-1)
possible_df = seq(2, 100, by=1) # list of all degrees of freedom we test (test from 3
degrees of freedom to 50)
mse_df = c(rep(0,length(possible_df))) # we choose the degree of freedom with the sma
llest MSE
for (i in 1:length(possible_df)) { # we go through each degree of freedom
  predictions = c(rep(0,100)) # predictions store the prediction of yi for observatio
n xi from i = 1 to 100
  true_val = c(rep(0,100)) # true_val stores yi corresponding to observation xi from
i = 1 to 100
  for (j in 1:n) { # begin leave-one out cross validation
    validation_set <- observations[j, , drop = FALSE] # the jth row is the validation
_set
    true_val[j] = validation_set$y # we get the y value from the jth row
    training_set <- observations[-j, , drop = FALSE] # the training set is observatio
ns excluding the jth row
    y <- training_set$y
    x <- training_set$x
    cubic_spline <- lm(y ~ bs(x, degree = 3, df = possible_df[i]), data = training_se
t) # we train the cubic spline on the training_set
    new_observation <- data.frame(x = validation_set$x, y = validation_set$y) # we fi
nd a prediction for yj given xj from the validation set
    predictions[j] = predict(cubic_spline, new_observation)
  }
  rss = sum((true_val - predictions)^2) # once we finish leave-one out cross validati
on for a given degree of freedom, we find the RSS of the cublc spline for this degree
of freedom
  mse_df[i] = 1/100 * (rss) # we find the MSE associated with this degree of freedom
}

x <- match(min(mse_df), mse_df) # we find the index of the minimum MSE
print(paste("A suitable integer value for the degrees of freedom is : " , possible_df
[x])) # we find the corresponding value of the degree of freedom corresponding to thi
s MSE
```

```
## [1] "A suitable integer value for the degrees of freedom is :  7"
```