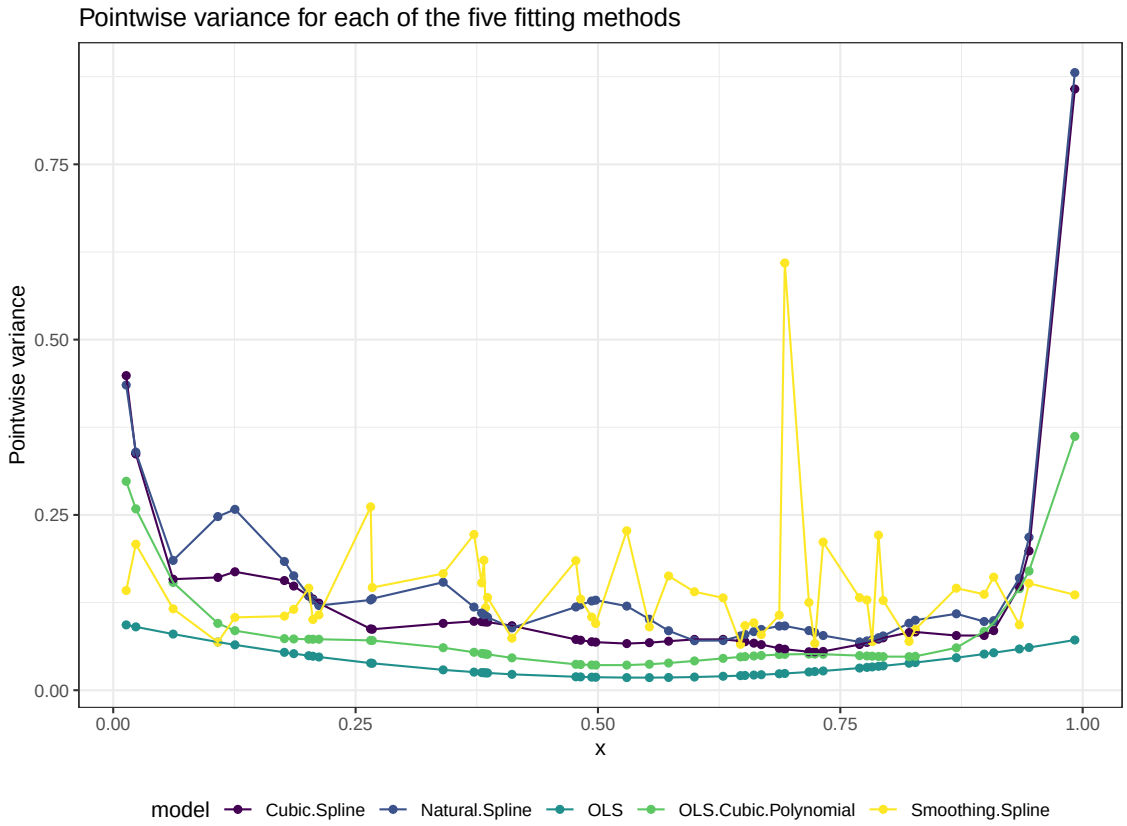
Homework 2

Ngoc Duong

10/16/2020

Problem 3 – Simulation

(Codes can be found in appendix)



Comment: Considering y is a complex function of x as given above, we see that OLS has consistently low pointwise variance (while having high bias), whereas there are high pointwise variances at the boundaries (0 and 1) for cubic splines. The natural cubic spline adds constraints on the function beyond the boundary, which may increase bias but in return decrease variance at these areas. The smoothing spline has relatively high pointwise variance across values of x, but the variances near the boundary points are controlled.

Problem 4

	Logistic.regression	LDA	QDA
Test error	0.2530864	0.2530864	0.2592593

	Logistic.regression	LDA	QDA
Test error SE	0.0342655	0.0342655	0.0345372

Comment: We can see that the test error rates as well as their standard errors over the test set for all three models are very similar. However, QDA seems to perform slightly less well on the test set. Since LDA makes strong assumptions about the distribution of the data and rely on the mean for discriminatory information, its performance might be affected if these assumptions are not met. Logistic regression can be a good choice due to its ability to estimate conditional probabilities for the observations as well as high intepretability. QDA might perform less well on the test set if the decision boundary is linear since it assumes a non-linear/quadratic decision boundary.

Appendix

Problem 3

```
#a) Generate a vector x consisting of 50 points drawn at random from Uniform[0,1]
set.seed(1)
x = runif(50, 0, 1)

#generate 100 training sets. Each training set consists of 50 pairs of (X; Y)
train_set = vector("list", 100)

#generate Y with formula given using x and e
for (i in 1:100){
y = (sin(2 * pi * x^3))^3 + rnorm(50, 0, 1)
train_set[[i]] = cbind(x = x, y = y)
}

ols.pred = NULL
#OLS with linear model and get fitted values
  for (i in 1:100){
  ols = lm(y ~ x, as.data.frame(train_set[[i]]))
  ols.pred = rbind(ols.pred, ols$fitted.values)
}

ols.cub.pred = NULL
#OLS with cubic term and get fitted values
for (i in 1:100){
  ols.cub = lm(y ~ x + I(x^2) + I(x^3), as.data.frame(train_set[[i]]))
  ols.cub.pred = rbind(ols.cub.pred, predict(ols.cub))
}

spline.cub.pred = NULL
#cubic spline with 2 knots at 0.33 and 0.66 and get fitted values
for (i in 1:100) {
  spline.cub = lm(y ~ bs(x, knots = c(0.33, 0.66)), as.data.frame(train_set[[i]]))
  spline.cub.pred = rbind(spline.cub.pred, predict(spline.cub))
}

nat.spline.pred = NULL
#natural cubic spline with 5 knots and get fitted values
for (i in 1:100){
```

```
  nat.spline = lm(y ~ ns(x, knots = c(0.1, 0.3, 0.5, 0.7, 0.9)), as.data.frame(train_set[[i]]))
  nat.spline.pred = rbind(nat.spline.pred, predict(nat.spline))
}

smooth.spline.pred = NULL
#smoothing spline with df selected by GCV and get fitted values from chosen model
for (i in 1:100){
  smoothspline = smooth.spline(x = as.data.frame(train_set[[i]])$x,
                               y = as.data.frame(train_set[[i]])$y, cv = TRUE)
  smooth.spline.pred = rbind(smooth.spline.pred, predict(smoothspline)$y)
}

#d) compute pointwise variance of fitted values
var = data.frame(x = x,
                 `OLS` = apply(ols.pred, 2, var),
                 `OLS Cubic Polynomial` = apply(ols.cub.pred, 2, var),
                 `Cubic Spline` = apply(spline.cub.pred, 2, var),
                 `Natural Spline` = apply(nat.spline.pred, 2, var),
                 `Smoothing Spline` = apply(smooth.spline.pred, 2, var))

#plot the pointwise variance curves against x for each model
var_long = pivot_longer(var, 2:6, names_to = "model", values_to = "variance")

var_long %>% group_by(model) %>% ggplot(aes(x = x, y = variance, col = model)) +
  geom_point() + geom_line() +
  labs(title = "Pointwise variance for each of the five fitting methods", y = "Pointwise variance") +
  theme_bw() +
  theme(legend.position = "bottom")
```

Problem 4

```
#import the data
sahd = read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.data", sep = ",", hea
  dplyr::select(-row.names) %>% mutate(chd = as.factor(chd))

#Divide the dataset into a training set consisting of the first 300 observations
train = sahd[1:300,]
test = sahd[301:462,]

#Apply logistic regression, LDA and QDA on the training set
#run logistic regression
log.mod = glm(chd ~., train, family = "binomial")
pred.log = predict(log.mod, test, type = "response")
pred.log = ifelse(pred.log > 0.5, 1, 0)

#run LDA
lda.mod = lda(chd ~., data = train)
pred.lda = predict(lda.mod, test)$class

#run QDA
qda.mod = qda(chd ~., data = train)
pred.qda = predict(qda.mod, test)$class

#report test error rate for each model
```

```r
error = data.frame(`Logistic regression` = mean(test$chd != pred.log),
             `LDA` = mean(test$chd != pred.lda),
             `QDA` = mean(test$chd != pred.qda))
rownames(error) = "Test error"

#compute standard error of the test error rate over the test set
err_se = data.frame(`Logistic regression` = sd(test$chd != pred.log)/ sqrt(nrow(test)),
                    `LDA` = sd(test$chd != pred.lda)/ sqrt(nrow(test)),
                    `QDA` = sd(test$chd != pred.qda)/sqrt(nrow(test)))
rownames(err_se) = "Test error SE"

#result
rbind(error, err_se) %>% knitr::kable()
```