P9120 Homework 2

Shan Jiang

October 16, 2019

Problem 1

Ex. 4.5

Consider a two-class logistic regression problem with $x \in R$. Characterize the maximum-likelihood estimates of the slope and intercept parameter if the sample x_i for the two classes are separated by a point $x_o \in R$. Generalize this result to (a) $x \in R_p$ and (b) more than two classes.

First we assume that $x_0 = 0$ and y = 1 for $x_i > 0$ and y = 0 for $x_i < 0$.

$$p(x;\beta) = \frac{\exp(\beta_x + \beta_0)}{1 + \exp(\beta_x + \beta_0)}$$

$$1 - p(x;\beta) = 1 - \frac{\exp(\beta_x + \beta_0)}{1 + \exp(\beta_x + \beta_0)} = \frac{1}{1 + exp(\beta_x + \beta_0)}$$

Given the condition that $x_0 = 0$ is the boundary, then $p(x_0) = 1 - p(x_0)$, $\beta_0 = 0$, so the above function can be simplified as:

$$p(x;\beta) = \frac{\exp(\beta_x)}{1 + \exp(\beta_x)}$$

$$1 - p(x;\beta) = 1 - \frac{1}{1 + \exp(\beta_x)}$$

Therefore we derive the likelihood function as:

$$L(\beta; y, x) = \prod_{i=1}^{N} p(x_i;\beta)^{y_i} [1 - p(x_i;\beta)]^{1-y_i}$$

$$= \prod_{i=1}^{N} [\frac{p(x_{i;}\beta)}{1 - p(x_{i;}\beta)}]^{y_i} [1 - p(x_i;\beta)]$$

$$= \prod_{i=1}^{N} [\exp(\beta x_i)]^{y_i} [1 - p(x_i;\beta)]$$

Then we take log of both sides:

$$logL(\beta; y, x) = \Sigma_{i=1}^{N} y_i[\beta x_i] - log[1 + exp\{\beta x_i\}]$$

After then we take the first-order derivatives w.r.t to β, and use the condition for y as an replacement.

$$\frac{dlogL(\beta; y, x)}{d\beta} = \Sigma_{i=1}^{N} x_i (y_i - \frac{exp(\beta x_i)}{1 + exp(\beta x_i)})$$

$$= \Sigma_{X_i>0}^{N} x_i(1 - \frac{exp(\beta x_i)}{1+exp(\beta x_i)}) - \Sigma_{x_i<0}^{N} x_i(\frac{exp(\beta x_i)}{1+exp(\beta x_i)})$$

$$= \Sigma_{X_i>0}^{N} x_i - \Sigma_{X_i>0}^{N} x_i(\frac{exp(\beta x_i)}{1+exp(\beta x_i)}) - \Sigma_{x_i<0}^{N} x_i(\frac{exp(\beta x_i)}{1+exp(\beta x_i)}).$$

$$set\frac{dlogL(\beta;y,x)}{d\beta} = 0 \Leftrightarrow \Sigma_{X_i>0}^{N} x_i = \Sigma_{x_i=1}^{N} x_i(\frac{exp(\beta x_i)}{1+exp(\beta x_i)})$$

Thus, we can infer that for any dataset $\Sigma_{x_i=1}^{N} x_i$, only when $\beta \to \infty$,the first order derivative can be set to 0 and holds.

(b) Suppose that there are K classes, with x_k sperates between K-1 class and K class, and -$\infty < x_0 < x_1 < x_2 < x_3 < \cdots < x_{k-1} < x_k = \infty$.

Each probability is defined as:

$$P_1(x;\beta) = \frac{exp(\beta_1 x + \beta_{01})}{1 + \Sigma_{j=1}^{K-1} exp(\beta_j x + \beta_{0j})}$$

$$P_2(x;\beta) = \frac{exp(\beta_2 x + \beta_{02})}{1 + \Sigma_{j=1}^{K-1} exp(\beta_j x + \beta_{0j})}$$

$$\vdots$$

$$P_{k-1}(x;\beta) = \frac{1}{1 + \Sigma_{j=1}^{K-1} exp(\beta_j x + \beta_{0j})}$$

Given the assumption, for y is y_i= 1 if $x_{j-1} < x_i < x_j$ and $y_i = 0$ otherwise for observation i = 1, . . . , N and class j = 1, . . . , K. we can take a new likelihood function:

$$L(\beta;y,x) = \prod_{j=1}^{K}\prod_{i=1}^{N_j}[p_j(x_i;\beta)^{y_i}]^{y_i},$$

where N_j is the number of obs. in class j, then we take the log-likelihood function:

$$logL(\beta;y,x) = \Sigma_{j=1}^{K-1}\Sigma_{i=1}^{N_j} log[\frac{exp(\beta_j x_i + \beta_{0j})}{1 + \Sigma_{j=1}^{K-1} exp(\beta_j x + \beta_{0j})}] + \Sigma_{i=1}^{N_k} y_i log[\frac{1}{1 + \Sigma_{j=1}^{k-1} exp(\beta_j x_i + \beta_{0j})}]$$

$$= \Sigma_{j=1}^{K-1}\Sigma_{i=1}^{N_j} y_i[\beta_j x_i + \beta_{0j}] - \Sigma_{j=1}^{K}\Sigma_{i=1}^{N_j} y_i log[1 + \Sigma_{j=1}^{k-1} exp(\beta_j x_i + \beta_{0j})]$$

Then we take the derivative wrt to $\beta = (\beta_1, \beta_2, \cdots, \beta_{k-1})$, given that for $x_{j-1} < x < x_j$, $p(x;\beta_j) = \frac{exp(\beta_j x + \beta_{0j})}{1+\Sigma_{j=1}^{k-1} exp(\beta_j x_i + \beta_{0j})}$, where $\beta_{0j} = log[exp(\beta_j x_{j-1}) - exp(\beta_j x_j)]$

The first-order derivative is then written as:

$$\frac{dlogL(\beta;y,x)}{d\beta_j} = \Sigma_{i=1}^{N_j} x_i + \Sigma_{i=1}^{N_j}\frac{exp(\beta_j x_{j-1})x_{j-1} - exp(\beta_j x_j)x_j}{exp(\beta_j x_{j-1}) - exp(\beta_j x_j)}$$

$$-\Sigma_{i=1}^{N}[X_i + \frac{exp(\beta_j x_{j-1} - exp(\beta_j x_j)x_j}{exp(\beta_j x_{j-1}) - exp(\beta_j x_j)}](\frac{exp(\beta_j x_i - \beta_{0j})}{1 + \Sigma_{j=1}^{K-1} exp(\beta_j x_i + \beta_{0j})})$$

Set the $\frac{dlogL(\beta;y,x)}{d\beta_j} = 0$, Since the question listed two scenarios:

(a) Now, suppose that there are two classes in which x $\in R_p$. Suppose that $\mathbf{X_1}$ and $\mathbf{X_2}$ are two vectors, we have that $\boldsymbol{\beta}(\mathbf{X_1} - \mathbf{X_2}) = 0$.

$$p(\mathbf{x};\beta) = \frac{exp(\beta' x + \beta_0)}{1 + exp(\beta' x + \beta_0)}$$

As we know that the X_0 is the seperating parameter, we can simplify the above equation as:

$$1 - p(\mathbf{x};\beta) = \frac{1}{1 + exp((x - x_0))}$$

This is similar to the univariate case in that once taking derivatives of the log-likelihood function wrt to $\beta = (\beta_1, \beta_2, \cdots, \beta_{k-1})$, and setting them equal to zero.

In conclusion, the Generalized form is that when $||\beta|| \to \infty$, the maximum likelihood estimator is attained, note that β is a vector.

Problem 2

Ex. 5.1 Show that the truncated power basis functions in (5.3) represent a basis for a cubic spline with the two knots as indicated.

$$h_1(X) = 1, h_3(X) = X^2, h_5(X) = (X - \xi_1)^3_+$$

$$h_2(X) = X, h_4(X) = X^3, h_5(X) = (X - \xi_2)^3_+$$

The proof requires a fulfillment of Condition C_2 leads to C_1,

The C_1 requires truncated power basis functions; While for C_2, it required a collection of cubic functions.

To prove from cubic function $C_2 \to C_1$

The cubic polynomial can be expressed by:

$$f(x) = \Sigma^6_{m=1} \beta_m h_m(x)$$

Then we need to show the continuity of the f(x) at knots $\xi_1$and ξ_2, and the first and second order derivatives' continuity.

1. The continuity of f(x):

By proving the left limit = right limit at ξ_1,the continuity can be achieved:

The Left-limit:

$$f(\xi_1 - k) = \beta_1 + \beta_2(\xi_1 - k) + \beta_3(\xi_1 - k)^2 + \beta_4(\xi_1 - k)^3 + \beta_5(\xi_1 - k - \xi_1)^3_+ + \beta_6(\xi_1 - k - \xi_2)^3_+, k > 0$$

$$= \beta_1 + \beta_2(\xi_1 - k) + \beta_3(\xi_1 - k)^2 + \beta_4(\xi_1 - k)^3 + 0 + 0$$

So,

$$lim_{k \to 0^-} f(\xi_1 - k) = \beta_1 + \beta_2 \xi_1 + \beta_3(\xi_1)^2 + \beta_4(\xi_1)^3$$

The right limit:

$$f(\xi_1 + k) = \beta_1 + \beta_2(\xi_1 + k) + \beta_3(\xi_1 + k)^2 + \beta_4(\xi_1 + k)^3 + \beta_5(\xi_1 + k - \xi_1)^3_+ + \beta_6(\xi_1 + k - \xi_2)^3_+, k > 0$$

$$lim_{k \to 0^+} f(\xi_1 + k) = \beta_1 + \beta_2 \xi_1 + \beta_3(\xi_1)^2 + \beta_4(\xi_1)^3$$

Thus, the left limit is equal to the right limit at ξ_1,the continuity can be achieved.

2. Continuity of $f'(x)$

Then we take the first order derivatives at $x = \xi_1$,

$$f'(\xi_1) = lim_{k \to 0} \frac{f(\xi_1) - f(\xi_1 - k)}{k}$$

Based on the definition of $f(x), and\ f(\xi_1) = \beta_1 + \beta_2\xi_1 + \beta_3(\xi_1)^2 + \beta_4(\xi_1)^3$ then we have:

$$f(\xi_1) - f(\xi_1 - k) = \beta_2\xi_1 + 2\beta_3(\xi_1)k + 3\beta_4(\xi_1)^2k + O(k)$$

Then the lfet-side derivative w.r.t. k is written as:

$$f'_{-}(\xi_1) = \beta_2\xi_1 + 2\beta_3(\xi_1) + 3\beta_4(\xi_1)^2$$

The right-side derivative is written as:

$$f'_{+}(\xi_1) = lim_{k \to 0} \frac{f(\xi_1 + k) - f(\xi_1)}{k}$$

For the same reason like above,

$$f(\xi_1 + k) - f(\xi_1) = \beta_2\xi_1 + 2\beta_3(\xi_1)k + 3\beta_4(\xi_1)^2k$$

$$+\beta_5(\xi_1 + k - \xi_1)^3_+ + \beta_6(\xi_1 + k - \xi_2)^3_+ + O(k)$$

$$f'_{+}(\xi_1) = \beta_2 + 2\beta_3\xi_1 + 3\beta_4\xi_1^2$$

3. Continuity of $f''(x)$

Then we take the second order derivatives at $x = \xi_1$

For the same reason like above part 2, that the left-side limit is equal to right-side limit,

$$f''(\xi_1) = f''_{-}(\xi_1) = f''_{+}(\xi_1) = 6\beta_4\xi_1^2$$

Conclusion: At ξ_1, function, first and second order derivatives are all continuous at this knot. Similarly, we can prove it on ξ_2, thus we can say the cubic spline function with two knots is equivalent to that of the basis function as it fufills the continuity requirement for function, first-order derivative and second-order derivative.

Problem 3

see next page R code.

Data Mining Homework 2

Shan Jiang

10/07/2019

Contents

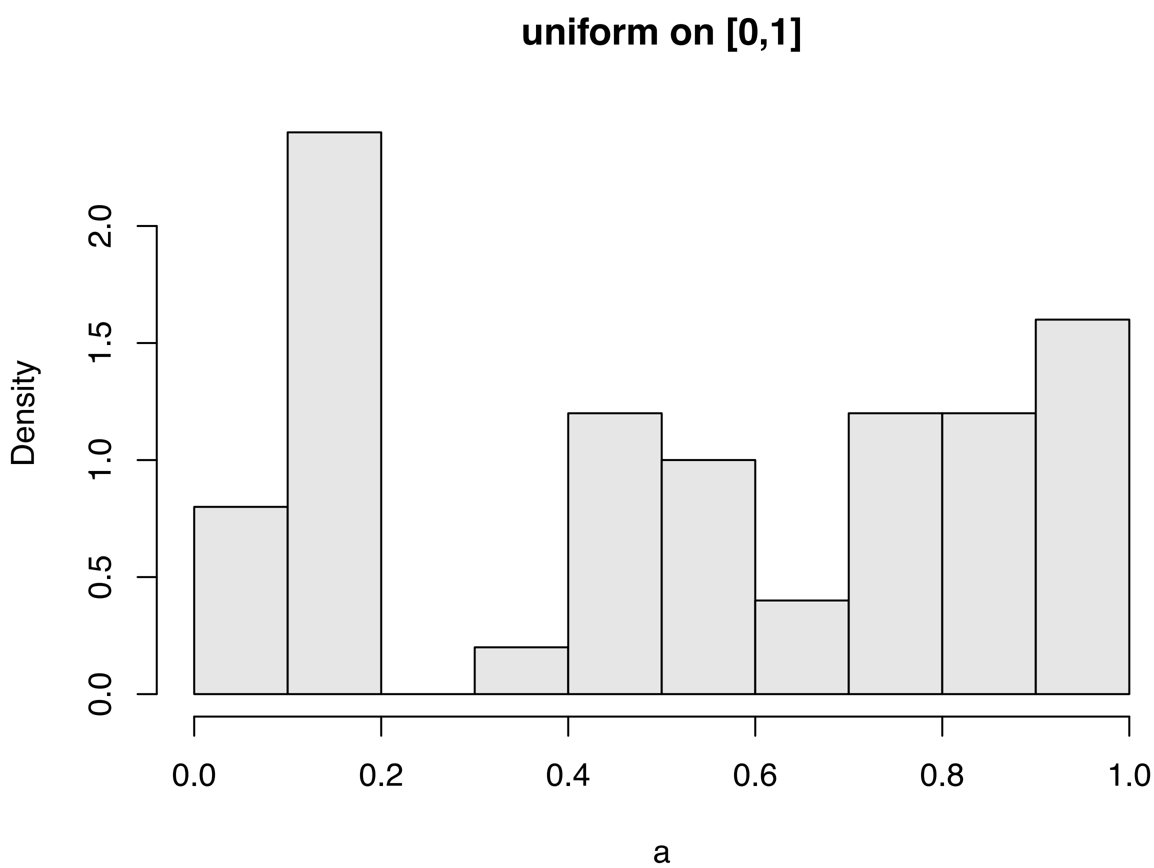
Problem 3

Load packages

```
library(tidyverse)
library(broom)
library(lattice)
library(caret)
library(glmnet)
library(MASS)
library(pls)
library(STAT)
library(splines)
require(ggplot2)
```

(a) Generate Vector x consisting of 50 points drawn at random from Uniform [0, 1].

```
## Basic prep
set.seed(1018)
a = as.vector(runif(50, 0,1))
hist(a,probability=TRUE,col=gray(.9),main="uniform on [0,1]")
```




```
## For each dataset X is identical
x.init = a
```

(b) Generate 100 training datasets

```r
n.sims <- 100
n.obs <- 50

# Make an empty list to save output in
xl = list()
el = list()
yl = list()
df = list()


## use for loop to iterate: 100 datasets
for (i in 1:n.sims){
  xl[[i]] = x.init
  el[[i]] = as.vector(rnorm(50, mean = 0, sd = 1)) ## std.normal
  yl[[i]] = (sin(2 * pi * (xl[[i]])^3 ))^3 + el[[i]]
  df[[i]] = data.frame(x = xl[[i]], y = yl[[i]])
}

## look at the data
#head(df[[100]])
#head(df[[50]])
```

(i) Data modelling

(i). OLS estimation

```r
set.seed(10008)

a1 = list()
b1 = list()

for (i in 1:n.sims){
  a1[[i]] = lm(y ~ x, df[[i]]) ## construct the model
  b1[[i]] = as.matrix(a1[[i]]$fitted.values) ## Fitted value list

}

## validation
fittest1 = lm(y~x,
       data = df[[100]])
tail(fittest1$fitted.values)
```

```
##         45         46         47         48         49         50
## 0.05177055 0.11952515 0.46370605 0.36409300 0.05419725 0.25554850
```

```r
tail(b1[[100]])
```

```
##          [,1]
## 45 0.05177055
```

```
## 46 0.11952515
## 47 0.46370605
## 48 0.36409300
## 49 0.05419725
## 50 0.25554850
```

The OLS linear model fitted values are stored in `b1[[i]]`.

(ii) OLS with cubic polynomial model

```
a2 = list()
b2 = list()

for (i in 1:n.sims){
  a2[[i]] = lm(y ~ poly(x, 3), df[[i]]) ## construct the model:cubic
  b2[[i]] = as.matrix(a2[[i]]$fitted.values) ## Fitted value list
}


## validation
fittest2 = lm(y~poly(x, 3),
       data = df[[100]])
tail(fittest2$fitted.values)
```

```
##           45           46           47           48           49
## -0.002370375  0.135360172  0.436949312  0.441226338  0.002810022
##           50
##  0.351611028
```

```
tail(b2[[100]])
```

```
##             [,1]
## 45 -0.002370375
## 46  0.135360172
## 47  0.436949312
## 48  0.441226338
## 49  0.002810022
## 50  0.351611028
```

$\beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$ cubic polynomial model, fitted values are stored in `b2[[i]]`.

(iii) Cubic Spline(B- Spline) with 2 knots

```
require(splines)
a3 = list()
b3 = list()

for (i in 1:n.sims){
  a3[[i]] = lm(y ~ bs(x,
                     knots = c(0.33,0.66)), df[[i]]) ## construct the model:cubic spline
  b3[[i]] = as.matrix(a3[[i]]$fitted.values) ## Fitted value list
```

```r
}

### Validate Model Coefficients
fittest3 <-lm(y ~ bs(x,
                    knots = c(0.33,0.66)),df[[100]])

#tail(fittest3$fitted.values)
#tail(b3[[100]])
```

fitted values are stored in `b3[[i]]`.

(iv) Natural Cubic Spline with 5 Knots

```r
set.seed(10008)
a4 = list()
b4 = list()

for (i in 1:n.sims){

  a4[[i]] = lm(y ~ ns(x,
                    knots = c(0.1, .3, .5, .7, .9)), df[[i]])

  b4[[i]] = as.matrix(a4[[i]]$fitted.values) ## Fitted value list
}

### Validate Model Coefficients
fittest4 <- lm(y ~ ns(x,
                    knots = c(0.1, .3, .5, .7, .9)),df[[100]])

tail(fittest4$fitted.values)
```

```
##          45          46          47          48          49          50
## -0.47151488  0.04837668  0.39564088  0.22643237 -0.50834441  0.42856132
```

```r
tail(b4[[100]])
```

```
##            [,1]
## 45 -0.47151488
## 46  0.04837668
## 47  0.39564088
## 48  0.22643237
## 49 -0.50834441
## 50  0.42856132
```

```r
# Plotting the data, the fit, and the 95% CI:
#plot(x, y, ylim = c(-1, +1))
#lines(df[[1]], b4[[1]], col = "darkred", lty = 2)
```

Fitted values are stored in `b4[[i]]`.

(v) Smoothing Spline with tuning parameter

The idea here is to transform the variables and add a linear combination of the variables using the Basis power function to the regression function f(x).

```
a5 = list()
b5 = list()

for (i in 1:n.sims){
   ## GCV choose tuning parameter
  a5[[i]] = smooth.spline(xl[[i]], yl[[i]],
                          cv = FALSE) ## Indicating GCV method
   ## Fitted value list
  b5[[i]] = as.matrix(a5[[i]]$y)
}

### Validate Model Coefficients
fittest5 <- smooth.spline(xl[[1]], yl[[1]],
                          cv = FALSE)
tail(fittest5$y)

## [1] -1.3747700 -1.0491052 -0.9801026 -0.8451319 -0.7764913 -0.4511079

tail(b5[[1]])

##              [,1]
## [45,] -1.3747700
## [46,] -1.0491052
## [47,] -0.9801026
## [48,] -0.8451319
## [49,] -0.7764913
## [50,] -0.4511079

### Plotting comparison
#plot(xl[[1]], yl[[1]],  col="grey",xlab="Xdf1",ylab="Ydf1")
#abline(v=c(0.1, .3, .5, .7, .9),lty= 2,col="darkgreen")
#lines(fittest5, col="red",lwd=2)
```

Fitted values are stored in `b5[[i]]`.

(c) Transform fitted value as dataframe

```
## Xij ith-variable, jth training set
xdf = as.data.frame(x.init)

### Extract fitted value and combine
data1list = list()
data2list = list()
data3list = list()
data4list = list()
data5list = list()
```

```r
for (i in 1:n.sims){
  data1list[[i]] = b1[i] %>%
   map_df(as_tibble)
}

fit_data1 = do.call(cbind, data1list)


for (i in 1:n.sims){
  data2list[[i]] = b2[i] %>%
            map_df(as_tibble)
}
fit_data2 = do.call(cbind, data2list)

for (i in 1:n.sims){
  data3list[[i]] = b3[i] %>%
      map_df(as_tibble)

}
fit_data3 = do.call(cbind, data3list)

for (i in 1:n.sims){
  data4list[[i]]  = b4[i] %>%
  map_df(as_tibble)

}

fit_data4 = do.call(cbind, data4list)

for (i in 1:n.sims){
  data5list[[i]] = b5[i] %>%
      map_df(as_tibble)

}
fit_data5 = do.call(cbind, data5list)

### Rename variable as Set and obs.
names(fit_data1) <- paste0("set",".", 1:100)
names(fit_data2) <- paste0("set",".", 1:100)
names(fit_data3) <- paste0("set",".", 1:100)
names(fit_data4) <- paste0("set",".", 1:100)
names(fit_data5) <- paste0("set",".", 1:100)

#head(fit_data5)
```

Thus we have simulated X_{ij} stored in dataframe `xdf`, while`fit_data1` `fit_data2`,…,`fit_data5` are storing values of $\hat{Y}_{ij}$ respectively from 5 models.

(d) Pointwise variance of Fitted values

```r
## Pointwise variance across 100 datasets
```

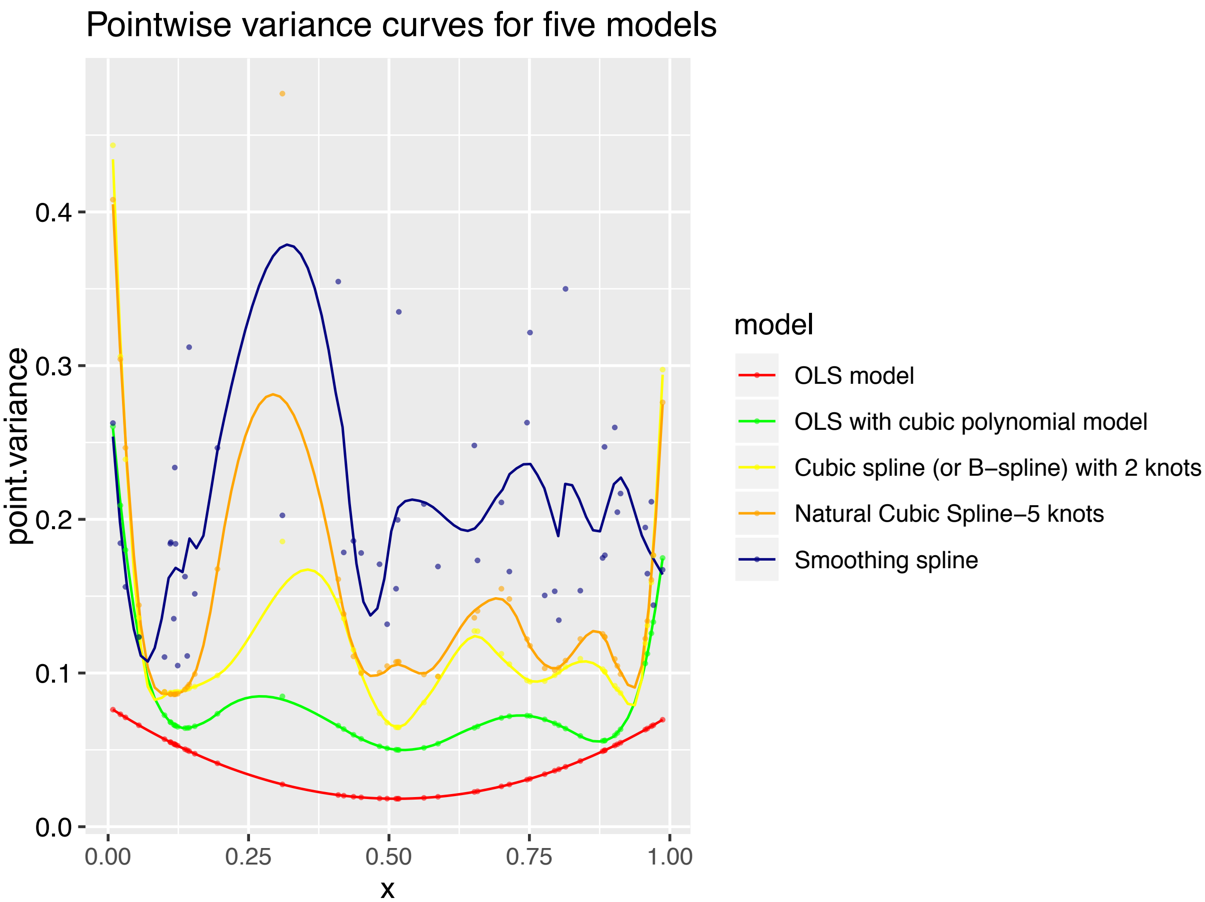
```r
pv1 = apply(fit_data1,1,var)
pv2 = apply(fit_data2,1,var)
pv3 = apply(fit_data3,1,var)
pv4 = apply(fit_data4,1,var)
pv5 = apply(fit_data5,1,var)
plotdf = as.data.frame(cbind(xdf, pv1, pv2, pv3, pv4, pv5))
#qplot(plotdf$x.init, plotdf$pv1, geom='smooth', span =0.1)
#qplot(plotdf$x.init, plotdf$pv2, geom='smooth', span =0.1)
```

Plotting

```r
colnames(plotdf) <- c("x", paste0("pv",".", 1:5))
plotdf = as.data.frame(plotdf)

plotdf = plotdf %>%
  gather("model", "point.variance", pv.1:pv.5)

ggplot(plotdf, aes(x = x, y = point.variance, col = model)) +
  geom_smooth(se=FALSE, method="loess", span=0.2, size=0.44) +
  geom_point(size = 0.33, alpha = 0.6) +
  ggtitle("pointwise variance") +
    theme(axis.text.y = element_text(colour = 'black', size = 10),
          axis.title.y = element_text(size = 12,
          hjust = 0.5, vjust = 0.2)) +
    theme(strip.text.y = element_text(size = 10, hjust = 0.5,
          vjust = 0.5, face = 'bold')) +
   labs(caption = "X consisting of 50 points drawn at random from U[0,1]", title ="Pointwise variance cu
    scale_color_manual(labels = c("OLS model",
                                  "OLS with cubic polynomial model",
                                  "Cubic spline (or B-spline) with 2 knots",
                                  "Natural Cubic Spline-5 knots",
                                  "Smoothing spline"),
                       values = c("red",
                                  "green", "Yellow", "orange","navy"))
```



X consisting of 50 points drawn at random from U[0,1]

Conclusion:

- The `global linear` model remains best in the variance across the range, including boundaries.
- `Cubic Polynomial`, `Natural Cubic` and `Cubic spline`both require a price paid in bias near the boundaries, see from the orange line in the figure.

Problem 4

South Africa data:

Data Description- Import data and Cleaning

```
## 'data.frame':    462 obs. of  10 variables:
##  $ sbp      : int  160 144 118 170 134 132 142 114 114 132 ...
##  $ tobacco  : num  12 0.01 0.08 7.5 13.6 6.2 4.05 4.08 0 0 ...
##  $ ldl      : num  5.73 4.41 3.48 6.41 3.5 6.47 3.38 4.59 3.83 5.8 ...
##  $ adiposity: num  23.1 28.6 32.3 38 27.8 ...
##  $ famhist  : num  1 0 1 1 1 1 0 1 1 1 ...
##  $ typea    : int  49 55 52 51 60 62 59 62 49 69 ...
##  $ obesity  : num  25.3 28.9 29.1 32 26 ...
##  $ alcohol  : num  97.2 2.06 3.81 24.26 57.34 ...
##  $ age      : int  52 63 46 58 49 45 38 58 29 53 ...
##  $ chd      : int  1 1 0 1 1 0 0 1 0 1 ...
```

```
## [1] 462  10
```

There are 10 variables in the data and 462 observations in total.

outcome (column 1): `chd` (response, coronary heart disease)

Predictors (columns 2–10)

- tobacco (cumulative tobacco (kg))
- ldl
- adiposity
- famhist
- typea (type-A behavior)
- obesity
- alcoho
- age
- sbp(systolic blood pressure)

Data split and normalization

As in the regression tutorial, we'll split our data into a training (first 300 observations) and testing (300-462 obs.) data sets, so we can assess how well our model performs on an out-of-sample data set.

Then we applied a normalization of predictor variables in the dataset.

```
## sampling segments
## Set up the train and test data
traindata = df[1:300,]
testdata = df[301:462,]

# standardization of predictors
trainst <- traindata
testst <- testdata

for(i in 1:9) {
  trainst[,i] <- trainst[,i] - mean(df[,i]);
  trainst[,i] <- trainst[,i]/sd(df[,i]);
}


for(i in 1:9) {
  testst[,i] <- testst[,i] - mean(df[,i]);
  testst[,i] <- testst[,i]/sd(df[,i]);
}
```

Data Analysis

For the Analysis below, all the data have been standardized already.

(1) logistic regression Model fit

Results from a logistic regression fit to the South African heart disease data.

(a) Model estimates and fit on train dataset

```
model1 <- glm(chd ~ .,
              family = "binomial",
              data = trainst)

tidy(model1)
```

```
## # A tibble: 10 x 5
##    term        estimate std.error statistic     p.value
##    <chr>          <dbl>     <dbl>     <dbl>       <dbl>
##  1 (Intercept)  -0.760      0.150    -5.07  0.000000399
##  2 sbp          -0.0981     0.154    -0.638 0.524
##  3 tobacco       0.318      0.148     2.14  0.0324
##  4 ldl           0.231      0.154     1.50  0.135
##  5 adiposity     0.363      0.290     1.25  0.211
##  6 famhist       0.408      0.138     2.95  0.00317
##  7 typea         0.461      0.156     2.95  0.00319
##  8 obesity      -0.267      0.224    -1.19  0.234
##  9 alcohol       0.125      0.145     0.862 0.388
## 10 age           0.572      0.214     2.68  0.00746
```

(b) Model prediction

```
# predictions
glm.probs <- round(predict(model1, testst,
                   type="response"))

# confusion matrix
table(testst$chd, ifelse(glm.probs > 0.5, 1, 0))
```

```
##
##      0  1
##   0 94 18
##   1 23 27
```

(c). Test Error and SE

```
# error rate ## 0.28
testst %>%
  summarise(logit.error = mean(ifelse(glm.probs > 0.5, 1, 0) != chd),
            logit.sd = sd(ifelse(glm.probs > 0.5, 1, 0) != chd))
```

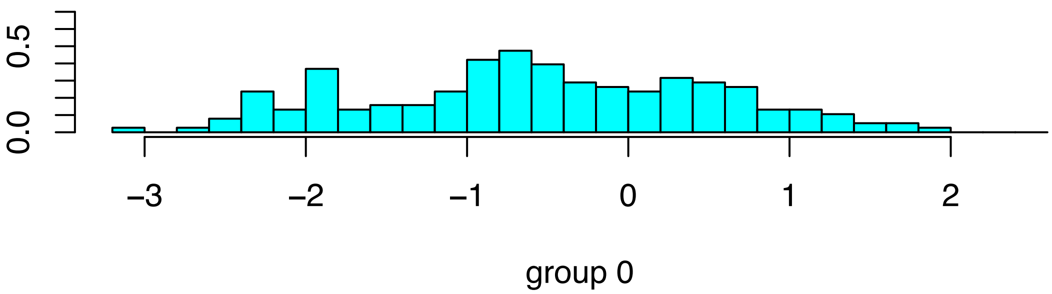
```
##   logit.error  logit.sd
## 1   0.2530864 0.4361282
```

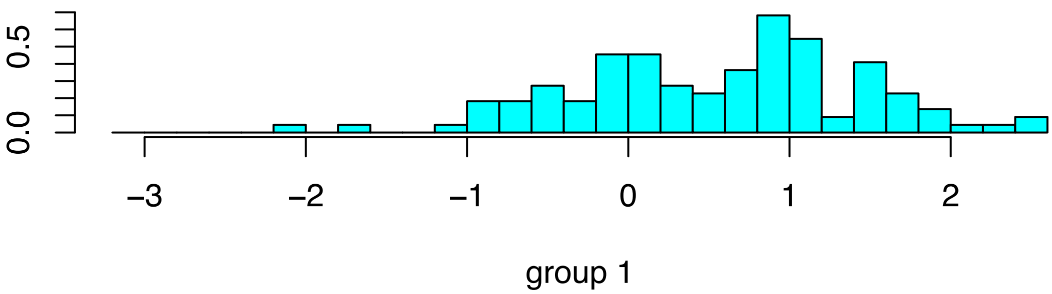
(2) LDA

LDA computes "discriminant scores" for each observation to classify what response variable class it is in (i.e. diseased or non-diseased).

(a). Model on trained data

```
lda.m1 <- lda(chd ~ .,
               data = trainst)
plot(lda.m1)
```





1. The LDA output indicates that our prior probabilities are $\pi_1 = 0.6333333, \pi_2 = 0.3666667$; in other words, 63.33% of the training observations are customers who did not have the heart disease and 36.67% represent those who are diseased.
2. It also provides the group means; these are the average of each predictor within each class, and are used by LDA as estimates of μ_k.
3. The coefficients suggest that subjects having a higher risk of getting the disease on average, are more likely to be smokers compared with non-diseased (-21% of non-diseased are smokers whereas 39.15% of diseased are).

(b). Predictions on Test data

```
## Fit on testst
test.predicted.lda <- predict(lda.m1, testst,
                              type = "response")

# number of high-risk patients with 50% probability of being diseased
sum(test.predicted.lda$posterior[, 2] > .5)
```

```
## [1] 47
```

```
## 2-2 Table classification: matrix
lda.cm <- table(testst$chd, test.predicted.lda$class)
list(LDA_model = lda.cm %>%
                   prop.table() %>%
                   round(3))
```

```
## $LDA_model
##
##         0     1
##   0 0.574 0.117
##   1 0.136 0.173
```

The default setting is to use a 50% threshold for the posterior probabilities.

(c). Test Error and SE

```
testst %>%
  mutate(lda.pred = test.predicted.lda$class) %>%
  summarise(lda.error = mean(chd != lda.pred),
            lda.se = sd(testst$chd != lda.pred))
```

```
##   lda.error    lda.se
## 1 0.2530864 0.4361282
```

(3) Quadratic discriminant analysis (QDA)

Quadratic discriminant analysis (QDA) provides an alternative approach. Like LDA, the QDA classifier assumes that the observations from each class of Y are drawn from a Gaussian distribution. However, unlike LDA, QDA assumes that each class has its own covariance matrix.

(a). Model estimates

```
qda.m1 <- qda(chd ~ ., data = trainst)
```

(b). Make Predictions

```
test.predicted.qda <- predict(qda.m1, newdata = testst)

## 2-2 Table classification
qda.cm <- table(testst$chd, test.predicted.qda$class)
list(QDA_model = qda.cm %>% prop.table() %>% round(3))
```

```
## $QDA_model
##
##         0     1
##   0 0.556 0.136
##   1 0.123 0.185
```

(c). Test Error and SE

```
testst %>%
  mutate(qda.pred = test.predicted.qda$class) %>%
  summarise(qda.error = mean(chd != qda.pred),
            qda.se = sd(testst$chd != qda.pred))
```

```
##   qda.error   qda.se
## 1 0.2592593 0.439587
```

Summary

```
require(knitr)
```

```
## Loading required package: knitr
```

```
m <- tibble( r0 = c( "Test Error", "SD"),
             r1 =      c(  0.2530864,  0.4361282),
             r2 =      c(  0.2530864,  0.4361282),
             r3 =     c(  0.2592593   ,   0.439587)  )
colnames(m) = c( "Model","Logistic", "LDA",  "QDA")
```

```
kable(m, digits = 5, align = "c",
              caption = "Summary test error and sd")
```

Table 1: Summary test error and sd

Model	Logistic	LDA	QDA
Test Error	0.25309	0.25309	0.25926
SD	0.43613	0.43613	0.43959

Comment:

- All three models give similar classification results.
- The test error and standard deviation are identical for logistic regression and LDA, which holds well as these two models are in this case are similar.
- While the QDA only differs a little bit as it's more complicated and has larger test error and standard deviation.