Solutions to Homework 4

Nakul Camasamudram

10/18/2017

Problem 1

a.

When $x \leq \xi$, $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. Comparing with $f(x)$, $a_1 = \beta_0$, $b_1 = \beta_1$, $c_1 = \beta_2$ and $d_1 = \beta_3$.

b.

For $x > \xi$,

$$\begin{aligned} f_2(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)_+^3 \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3x^2\xi + 3x\xi^2 - \xi^3) \\ &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2) x + (\beta_2 - 3\beta_4 \xi) x^2 + (\beta_3 + \beta_4) x^3 \end{aligned}$$

Hence, $a_2 = (\beta_0 - \beta_4 \xi^3)$, $b_2 = (\beta_1 + 3\beta_4 \xi^2)$, $c_1 = (\beta_2 - 3\beta_4 \xi)$ and $d_1 = (\beta_3 + \beta_4)$.

c.

$$\begin{aligned} f_2(\xi) &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2)\xi + (\beta_2 - 3\beta_4 \xi)\xi^2 + (\beta_3 + \beta_4)\xi^3 \\ &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + (-\beta_4 + 3\beta_4 - 3\beta_4 + \beta_3 + \beta_4)\xi^3 \\ &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3 \\ &= f_1(\xi) \end{aligned}$$

d.

From subproblem `c` above, $f_2(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3$. Then,

$$\begin{aligned} f_2^{'}(\xi) &= \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2 \\ &= f_1^{'}(\xi) \end{aligned}$$

e.

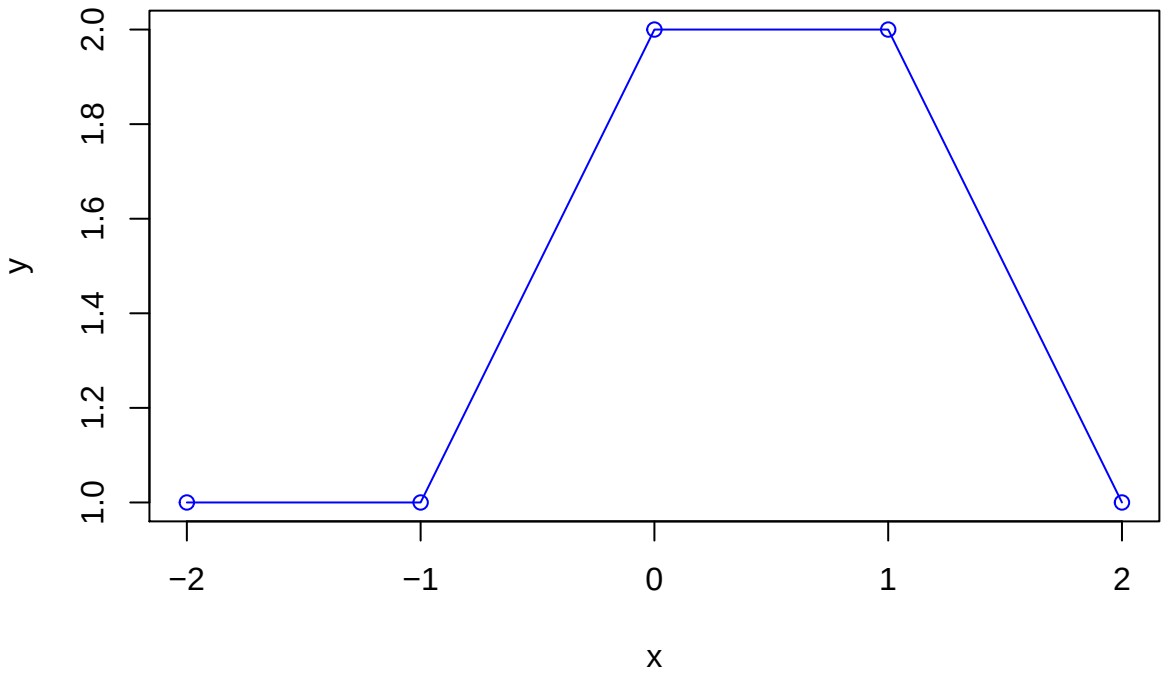
From subproblem `d` above, $f_2(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3$. Then,

$$\begin{aligned} f_2^{"}(\xi) &= 2\beta_2 + 6\beta_3 \\ &= f_1^{"}(\xi) \end{aligned}$$

Problem 2

x	y
-2	$1 + 0 + 0 = 1$
-1	$1 + 0 + 0 = 1$
0	$1 + 1 + 0 = 2$
1	$1 + (1\text{-}0) + 0 = 2$
2	$1 + (1\text{-}1) + 0 = 1$

```
x = -2:2
y = c(1 + 0 + 0,
      1 + 0 + 0,
      1 + 1 + 0,
      1 + (1-0) + 0,
      1 + (1-1) + 0)
plot(x, y, type = "o", col = "blue")
```



Problem 3

Based on the order of the derivate penalty functions, $\hat{g_2}$ is a higher order polynomial as compared to $\hat{g_1}$.

a.

$\hat{g_2}$ will have a smaller training RSS as it's order is higher than $\hat{g_1}$.

b.

$\hat{g_1}$ would have a smaller test RSS as $\hat{g_2}$ could overfit the training data.

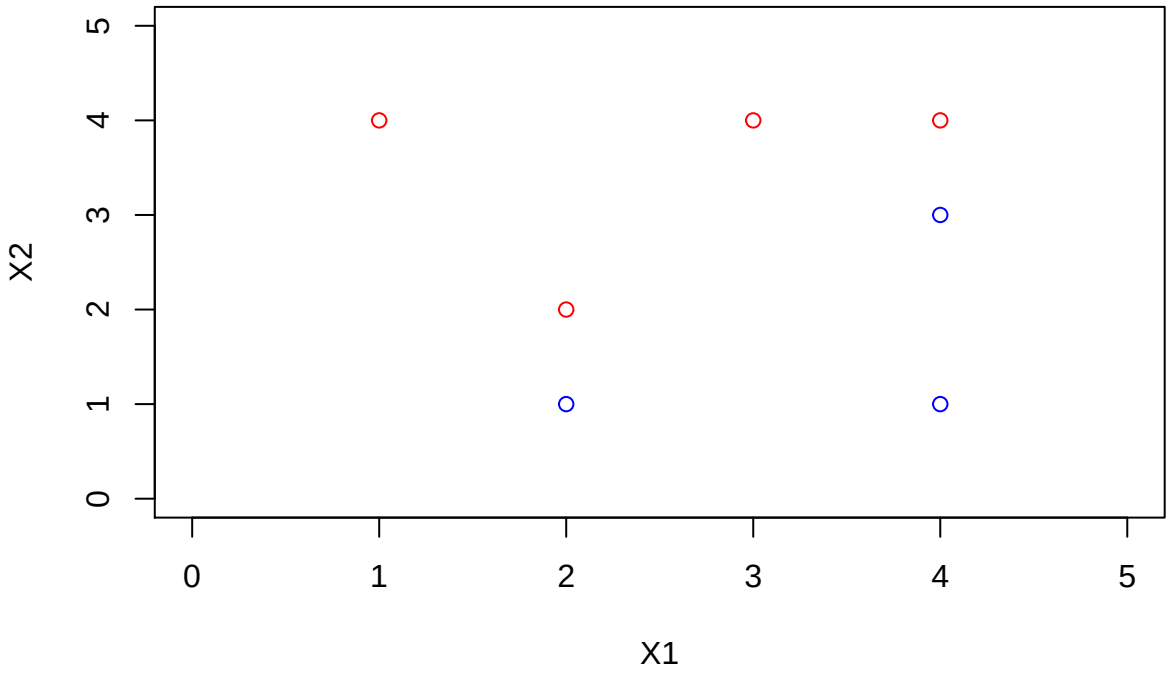
c.

For $\lambda = 0$, $\hat{g_1} = \hat{g_2}$. Hence, their training and test errors would be exactly the same.

Problem 4

a.

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
y = append(rep("red", 4), rep("blue", 3))
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
```

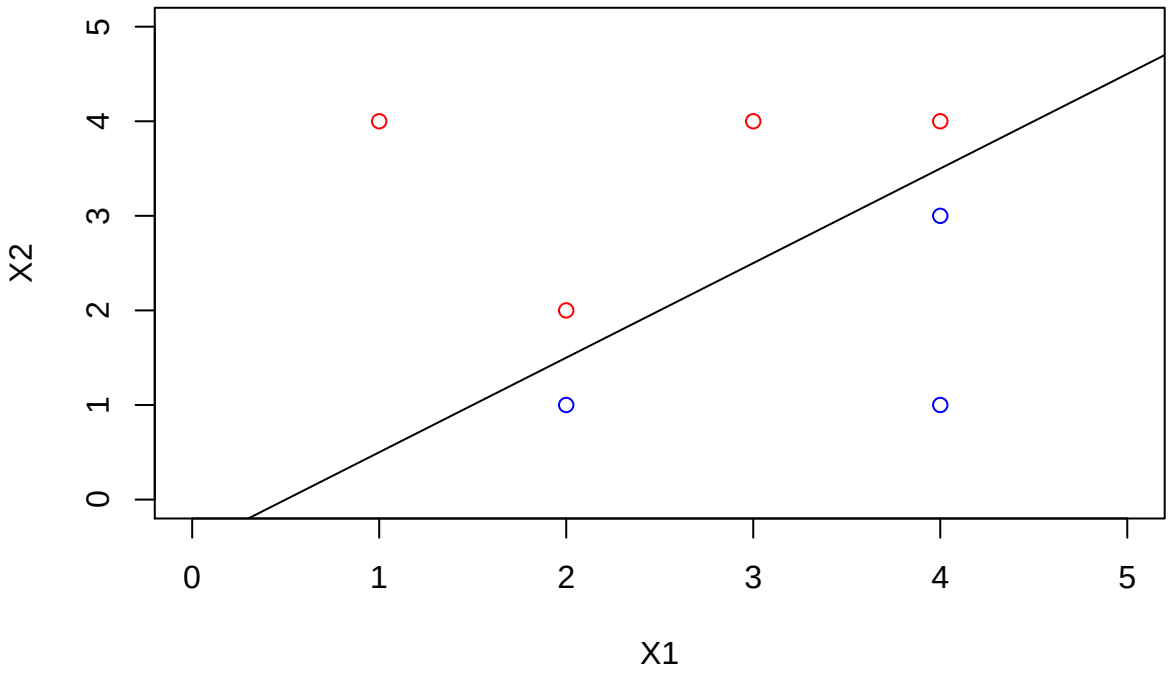


b.

The best margin classifier has to pass through the region between observations 2`(2, 2)`, 3`(4, 4)` and 5`(2, 1)`, 6`(4, 3)`. So, consider it passing through points `(2, 1.5)` and `(4, 3.5)`.

Slope = $\frac{3.5-1.5}{4-2} = \frac{2}{2} = 1$
Intercept = 3.5 - 4 = -0.5

```
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
abline(-0.5, 1)
```

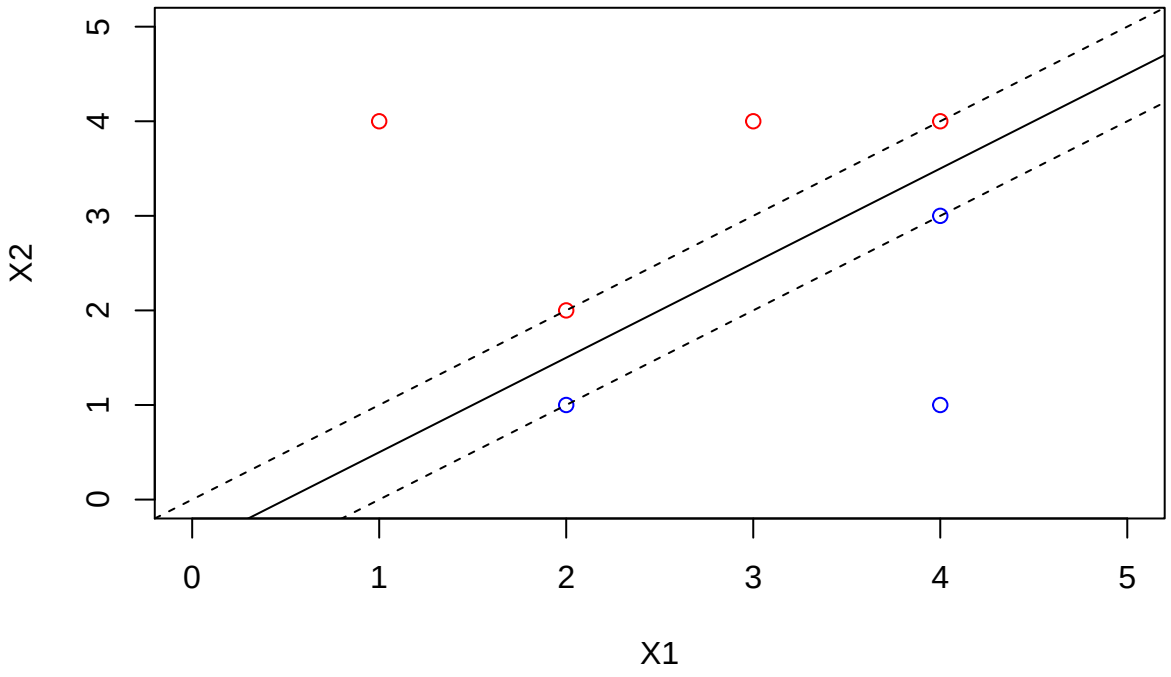


c.

Classify as `red` if X2 - X1 + 0.5 $>$ 0. Otherwise `blue`.

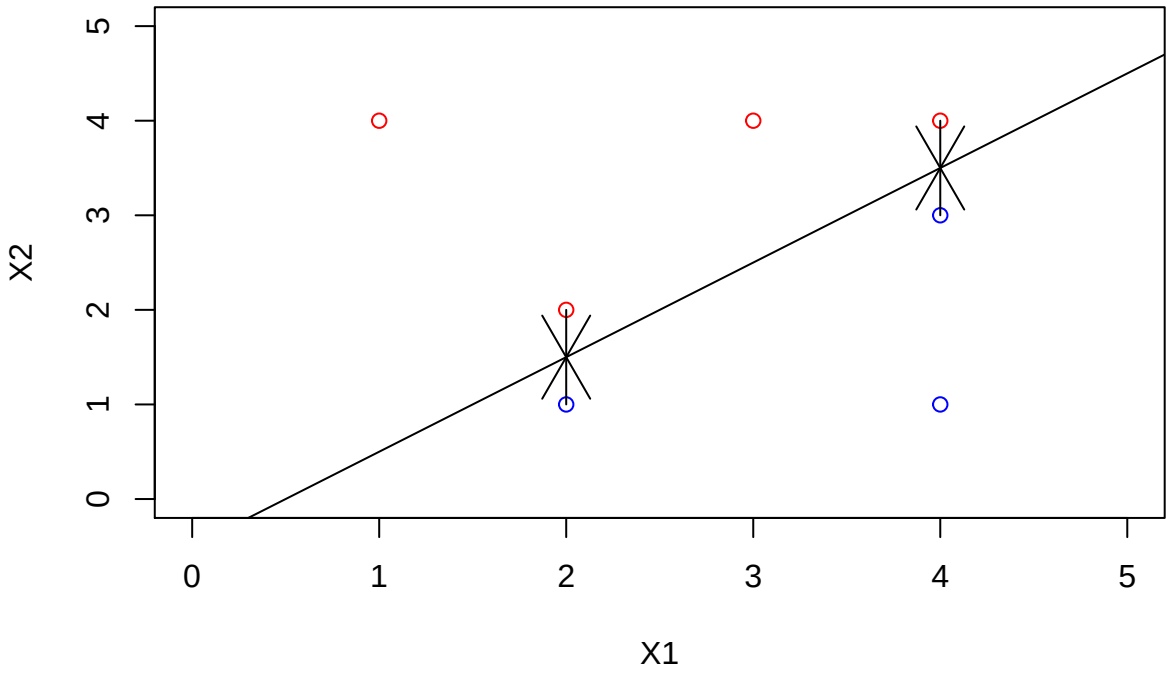
d.

```
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



e.

```
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
abline(-0.5, 1)
arrows(4, 4, 4, 3.5)
arrows(4, 3, 4, 3.5)
arrows(2, 1, 2, 1.5)
arrows(2, 2, 2, 1.5)
```

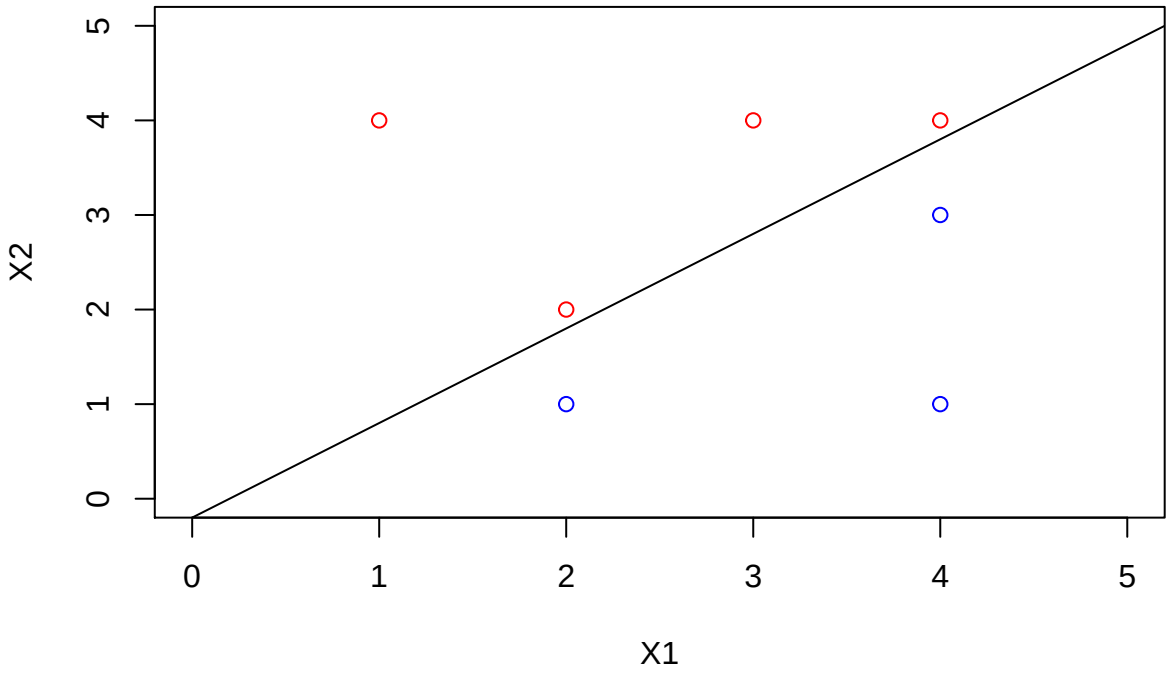


f.

Since observation 7 is outside the margin, it's movement should not affect the maximal margin hyperplane.

g.

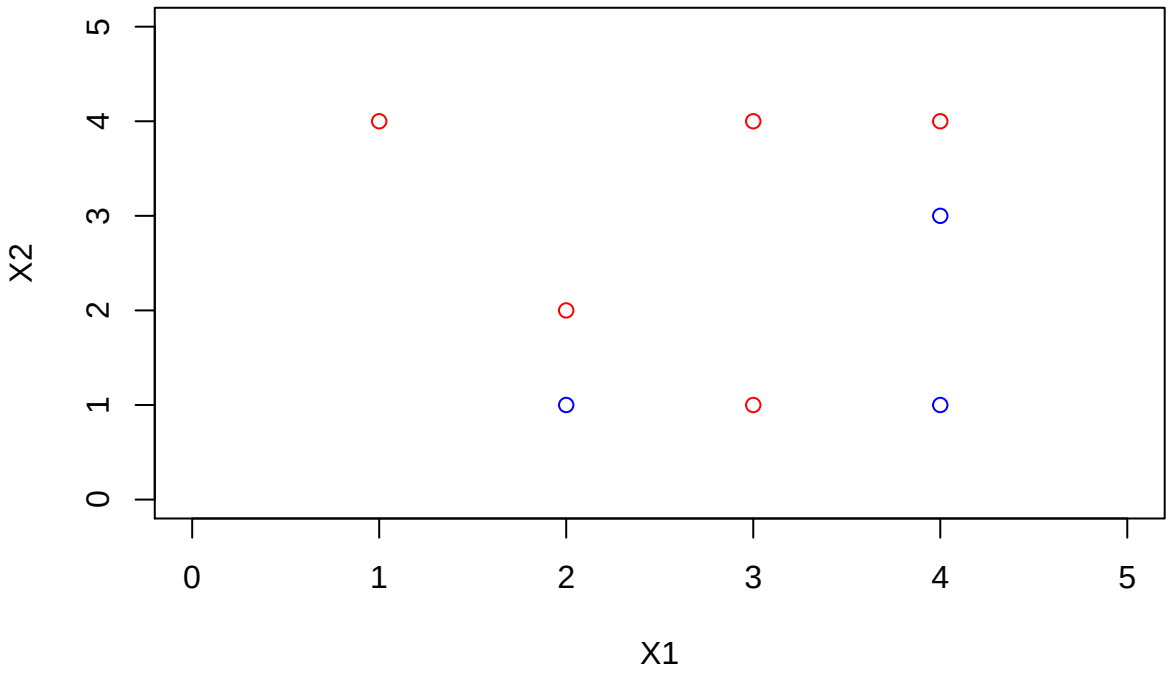
```
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
abline(-0.2, 1)
```



h.

Adding a point at `(3, 1)` would make the dataset linearly inseparable.

```
x1 <- c(x1, 3)
x2 <- c(x2, 1)
y <- c(y, "red")
plot(x1, x2, col = y, xlim = c(0, 5), ylim = c(0, 5), xlab = "X1", ylab = "X2")
```


X2
0
1
2
3
4
5
X1
0
1
2
3
4
5

Problem 5

Data Initialization

```
# Imports
library(dplyr)

rm(list=ls())
set.seed(123)

# Read the data
saheart <- read.table("SAheart.txt", sep = ",", header = TRUE) %>% dplyr::select(-row.names)

dim(saheart)
```

```
> [1] 462  10
```

```
levels(saheart$famhist)
```

```
> [1] "Absent"  "Present"
```

There are 462 rows in the dataset. Let's separate them equally into training and validation sets. Also, `famhist` is categorical with values "Absent" and "Present". Let's represent them as "0" and "1" respectively.

```
# Categorial to numeric
levels(saheart$famhist) <- c(0, 1)
saheart$famhist <- as.character(saheart$famhist)
saheart$famhist <- as.numeric(saheart$famhist)
saheart$chd <- as.factor(saheart$chd)

# Split dataset
ratio <- sample(1:nrow(saheart), 231)
saheart_training <- saheart[ratio, ]
saheart_validation <- saheart[-ratio, ]
```

a and b.

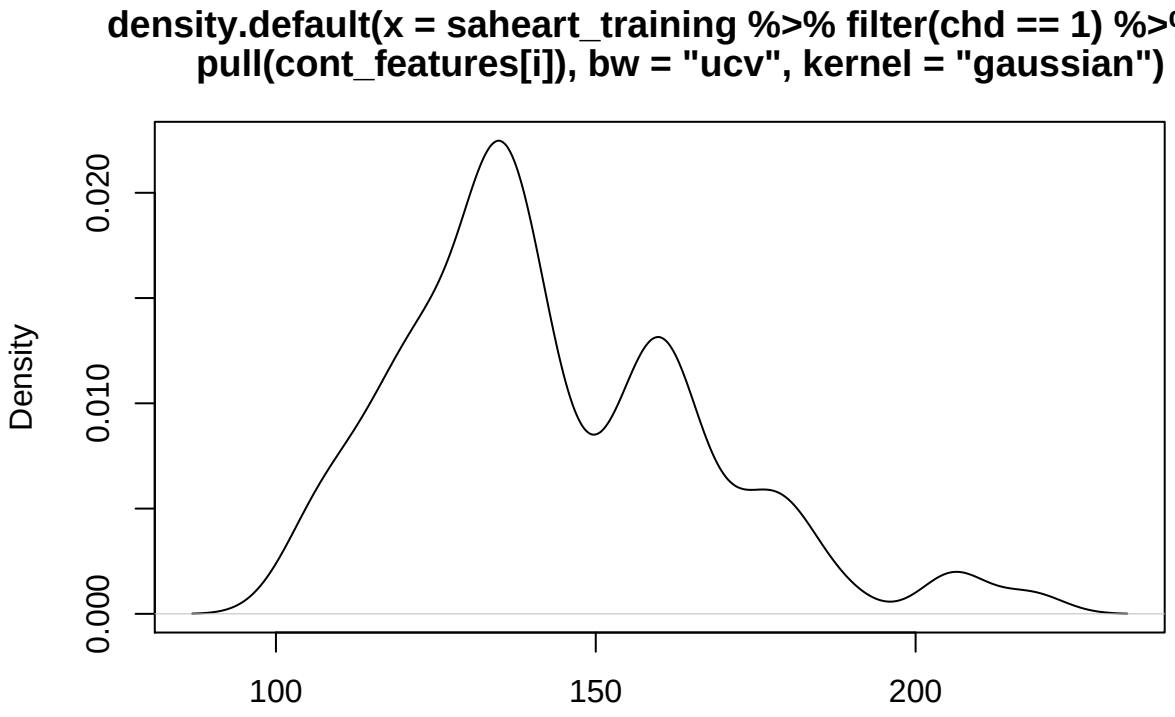
Let's generate the densities for each class and predictor in advance; using cross validation to select the best bandwidth parameter λ.

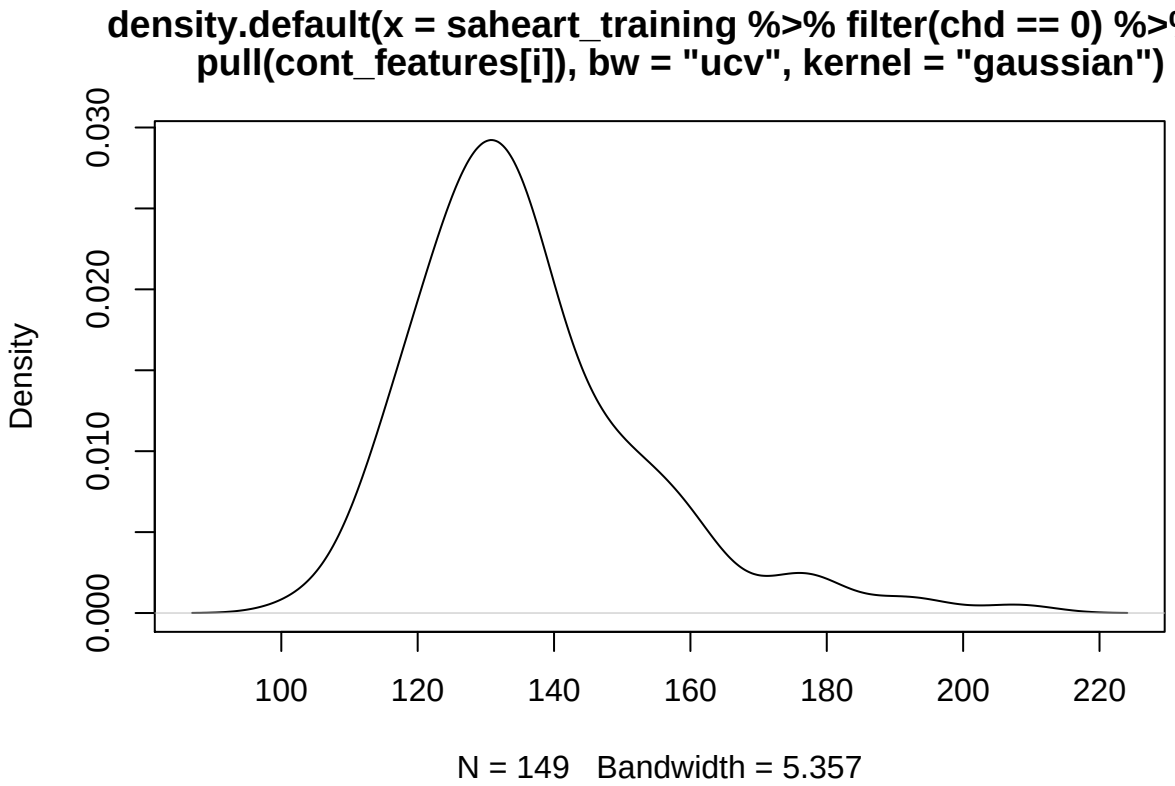
```
cont_features <- names(saheart_training %>% dplyr::select(-famhist, -chd))

densities.zero <- list()
densities.one <- list()
for(i in 1:length(cont_features)) {
    densities.zero[[i]] <- density(saheart_training %>% filter(chd==0) %>% pull(cont_features[i]),
                                   kernel = "gaussian", bw = "ucv")
    densities.one[[i]] <- density(saheart_training %>% filter(chd==1) %>% pull(cont_features[i]),
                                   kernel = "gaussian", bw = "ucv")
}
```

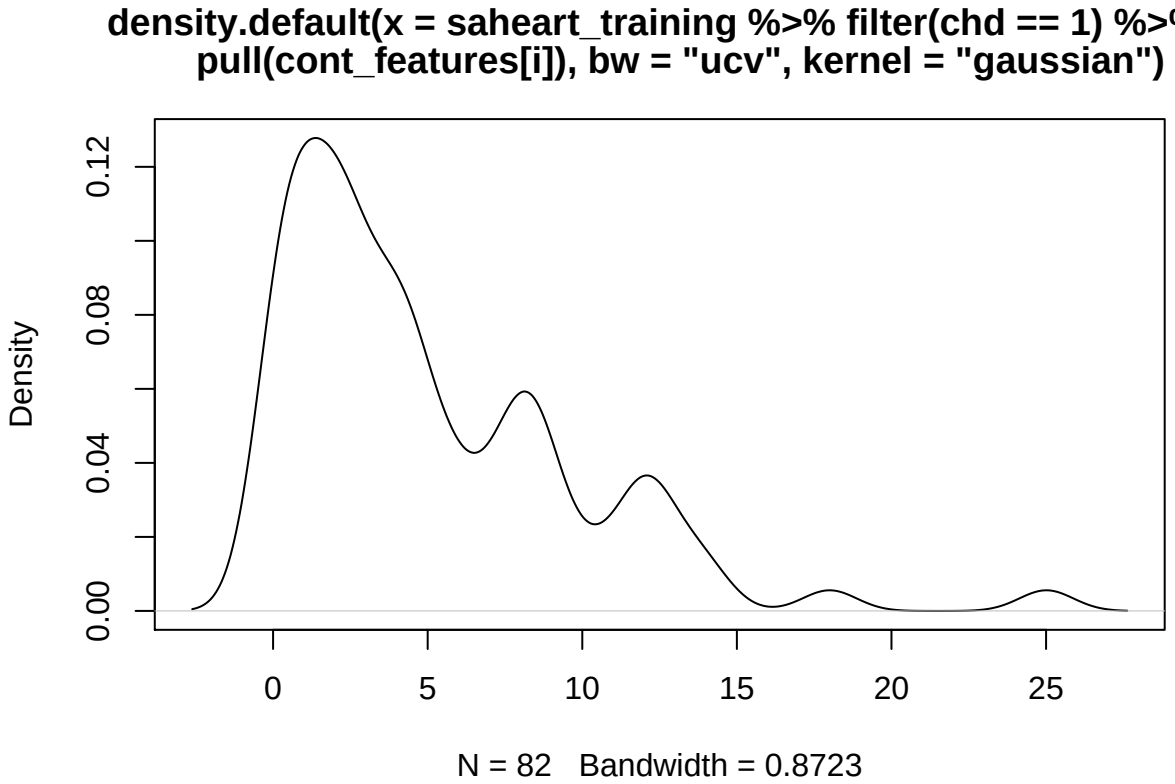
Since the densities are computed, let's visualize them and look at the bandwidth parameters

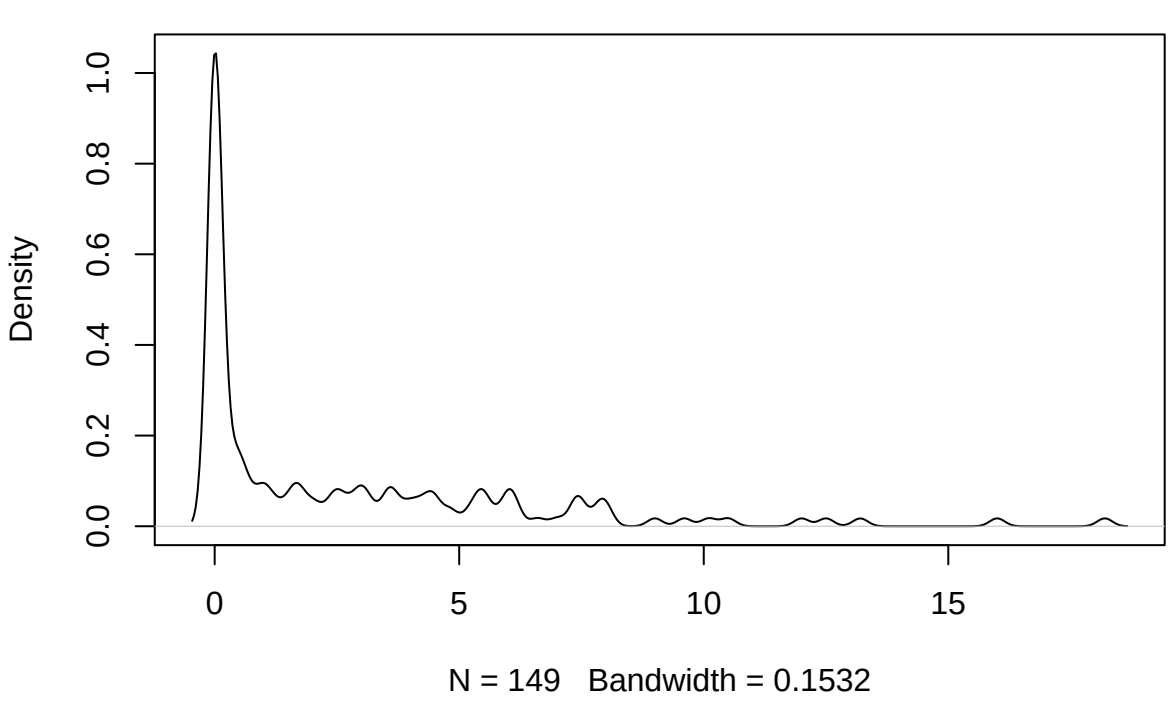
```
for(i in 1:length(cont_features)) {
  plot(densities.one[[i]])
```

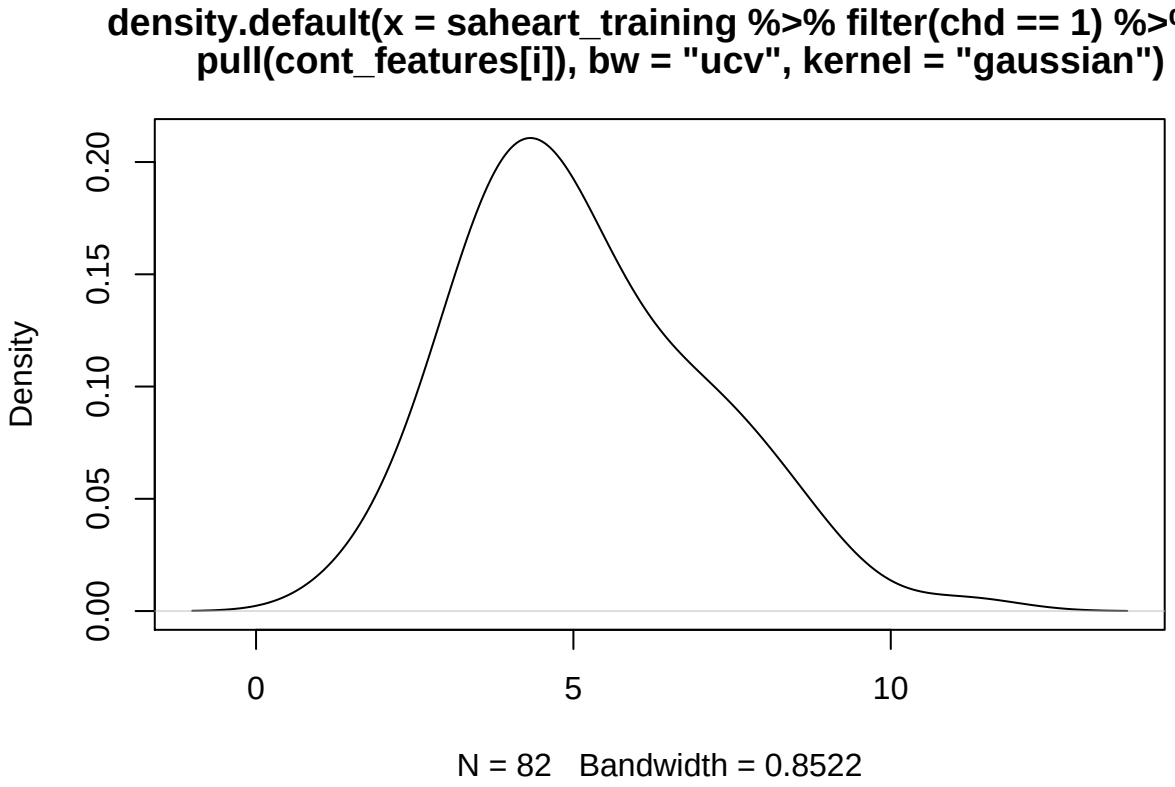
```
  plot(densities.zero[[i]])
}
```

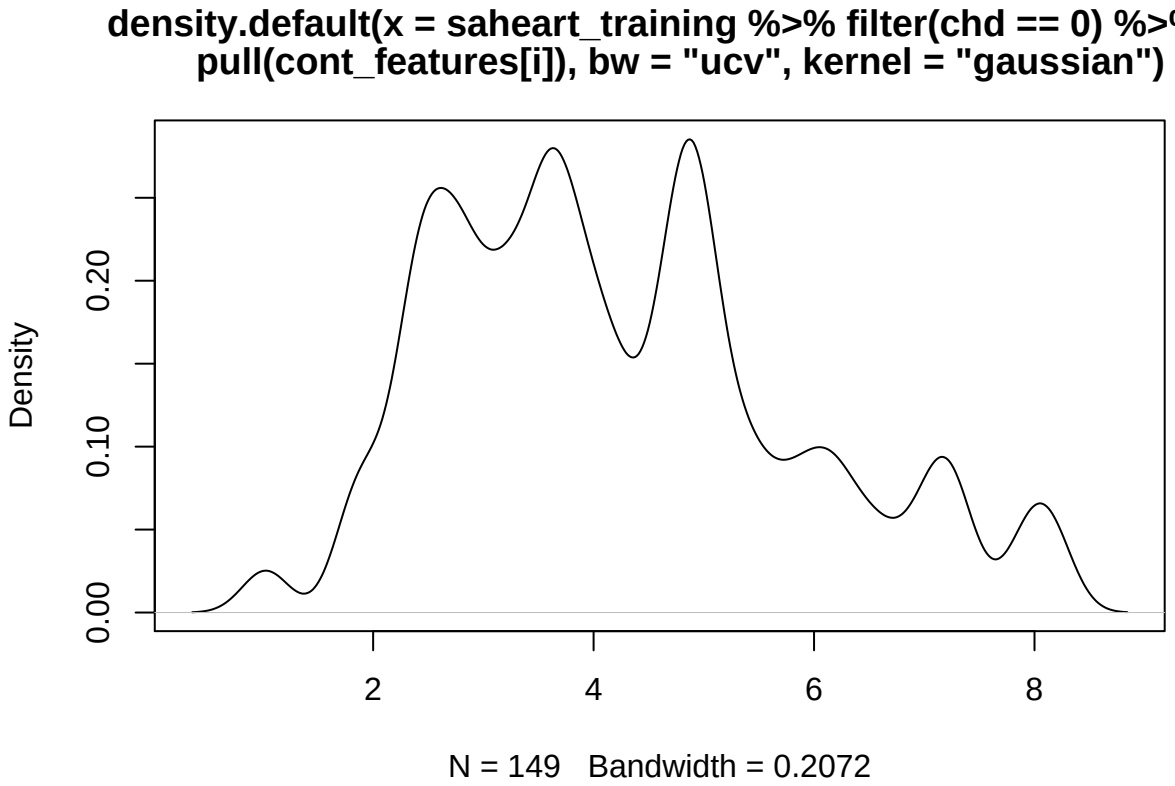


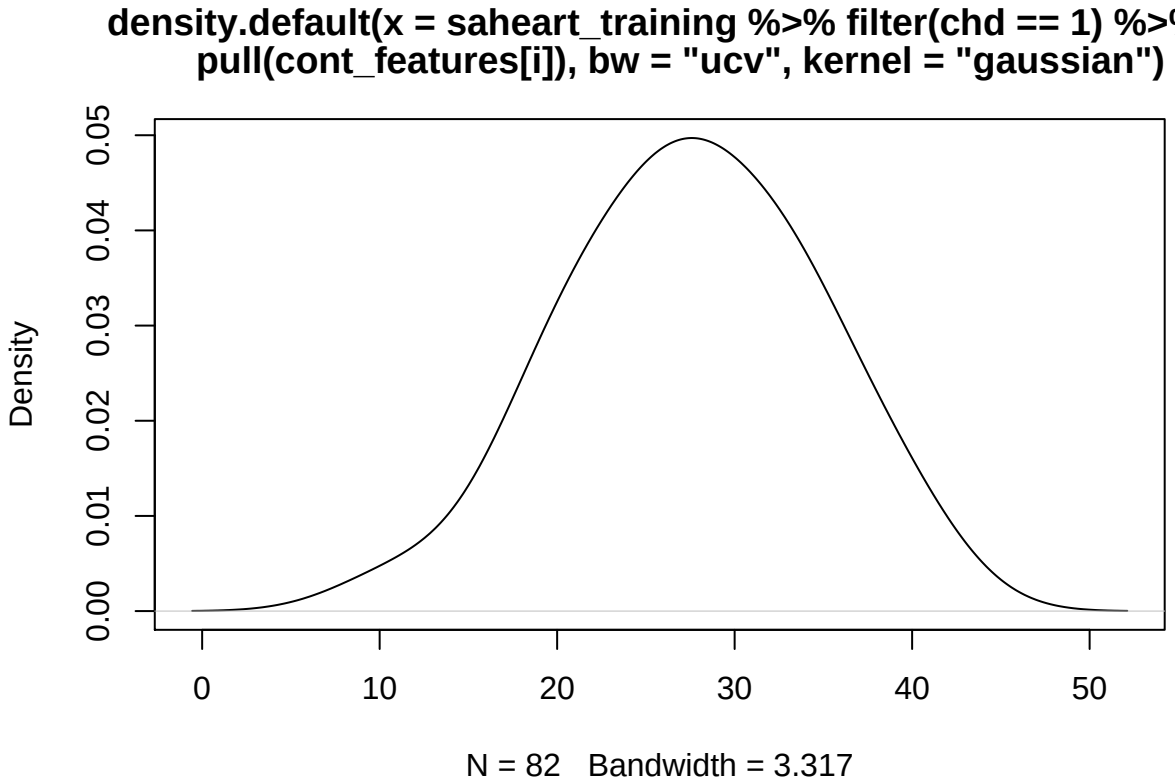


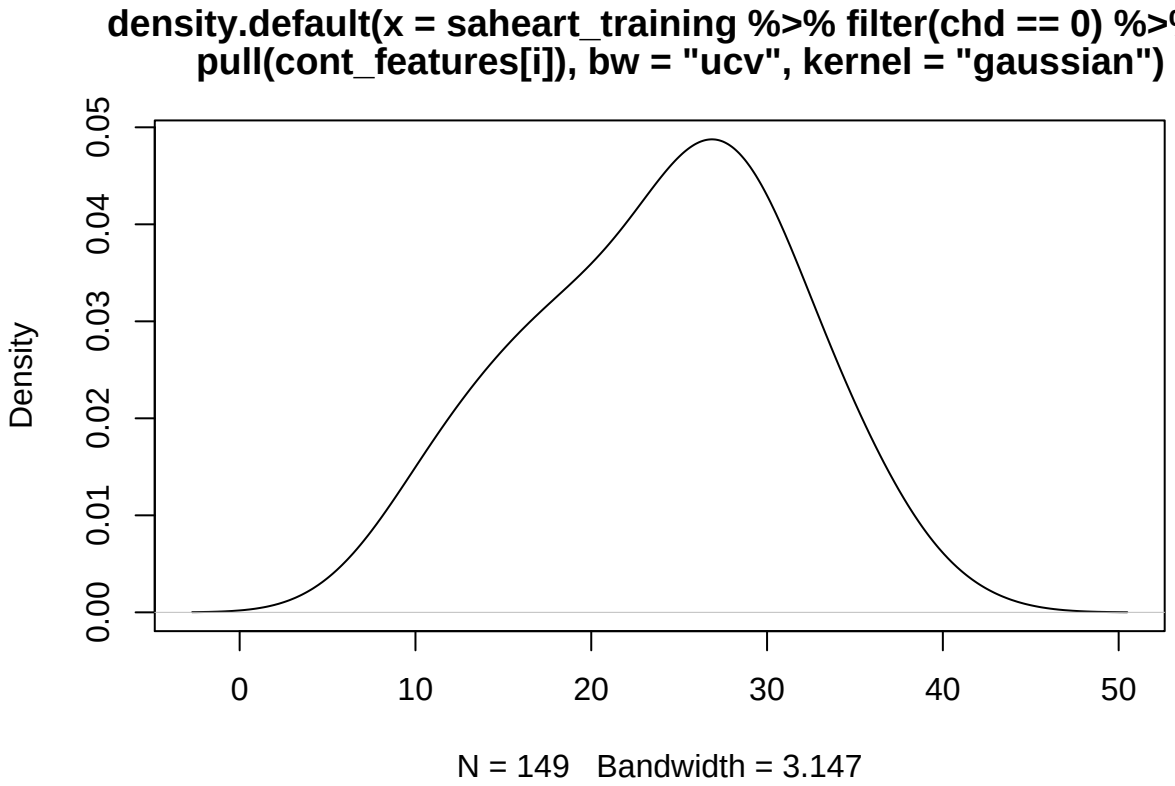

density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.04
0.08
0.12
0
5
10
15
20
25
N = 82 Bandwidth = 0.8723

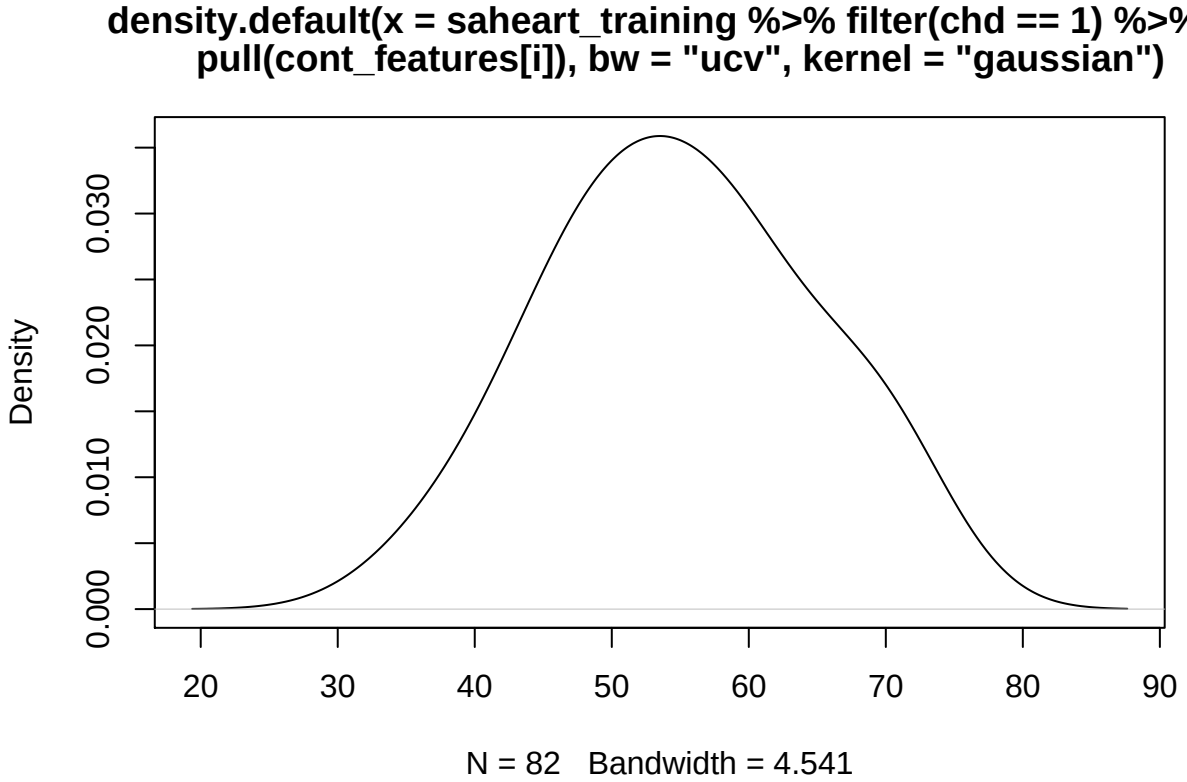

density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
N = 149 Bandwidth = 0.1532

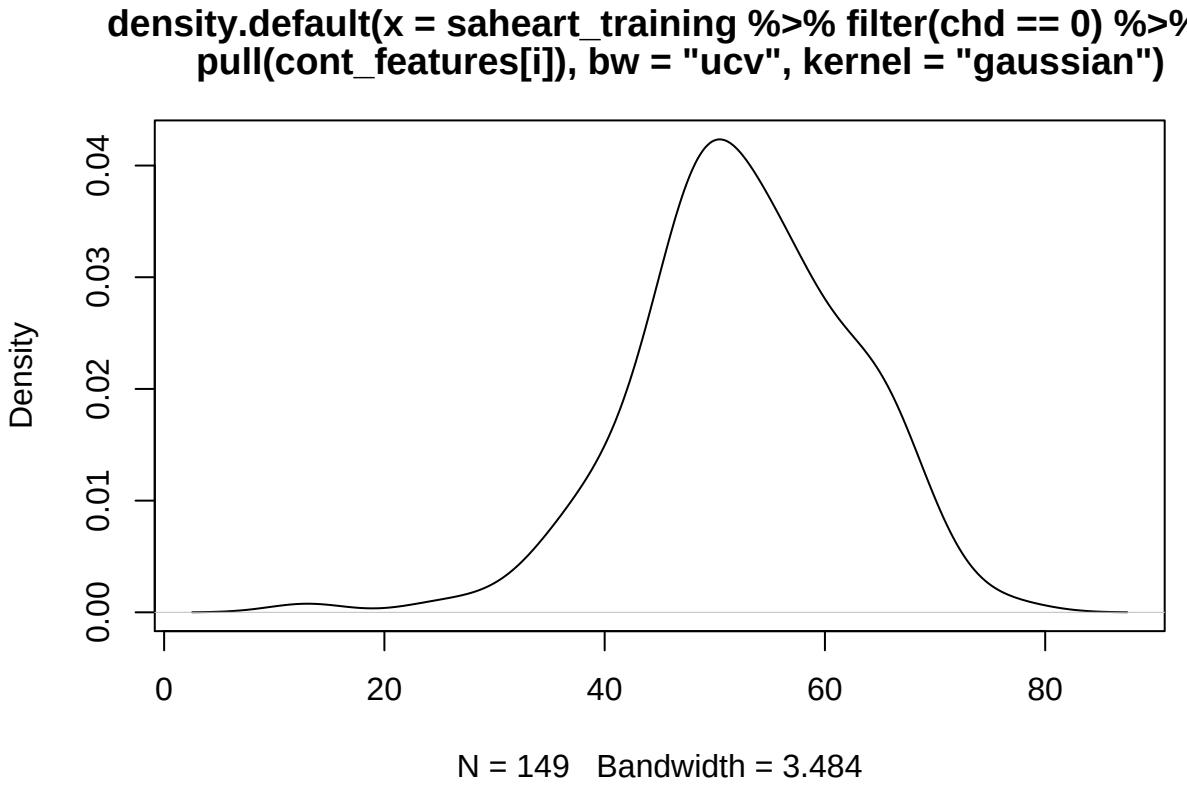

density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.05
0.10
0.15
0.20
0
5
10
N = 82 Bandwidth = 0.8522

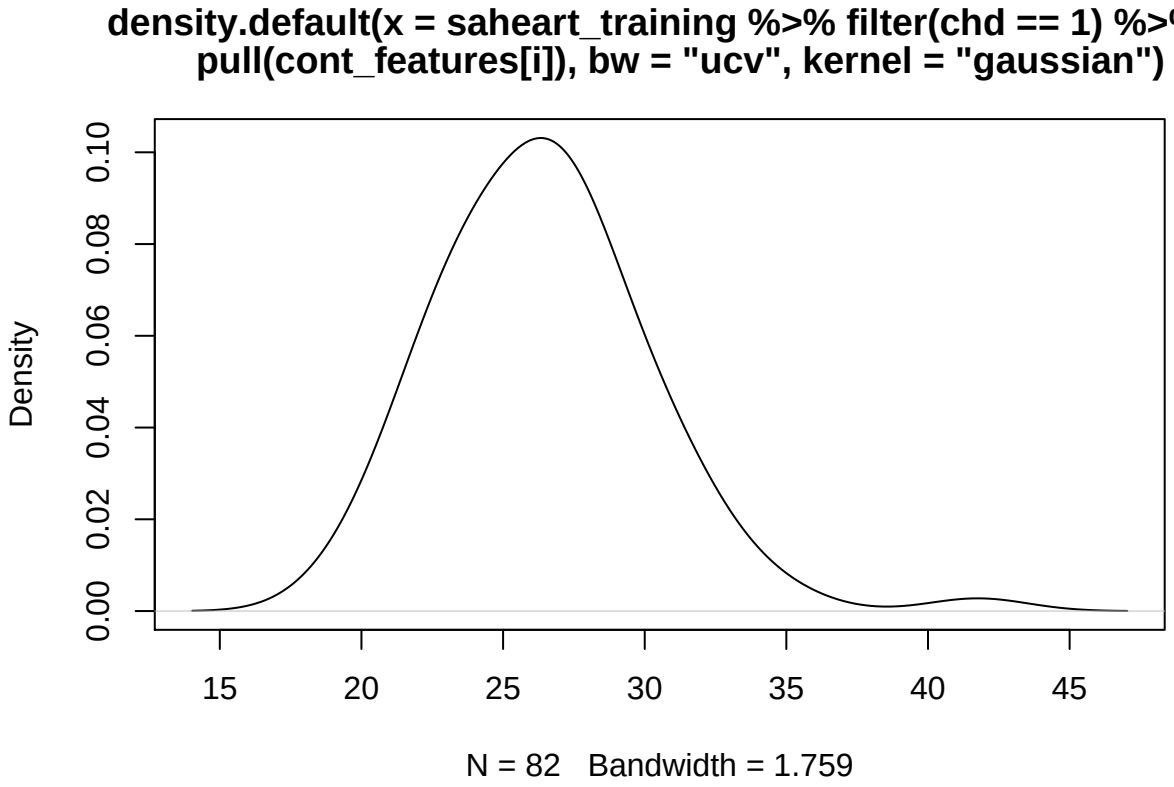

density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.10
0.20
2
4
6
8
N = 149 Bandwidth = 0.2072

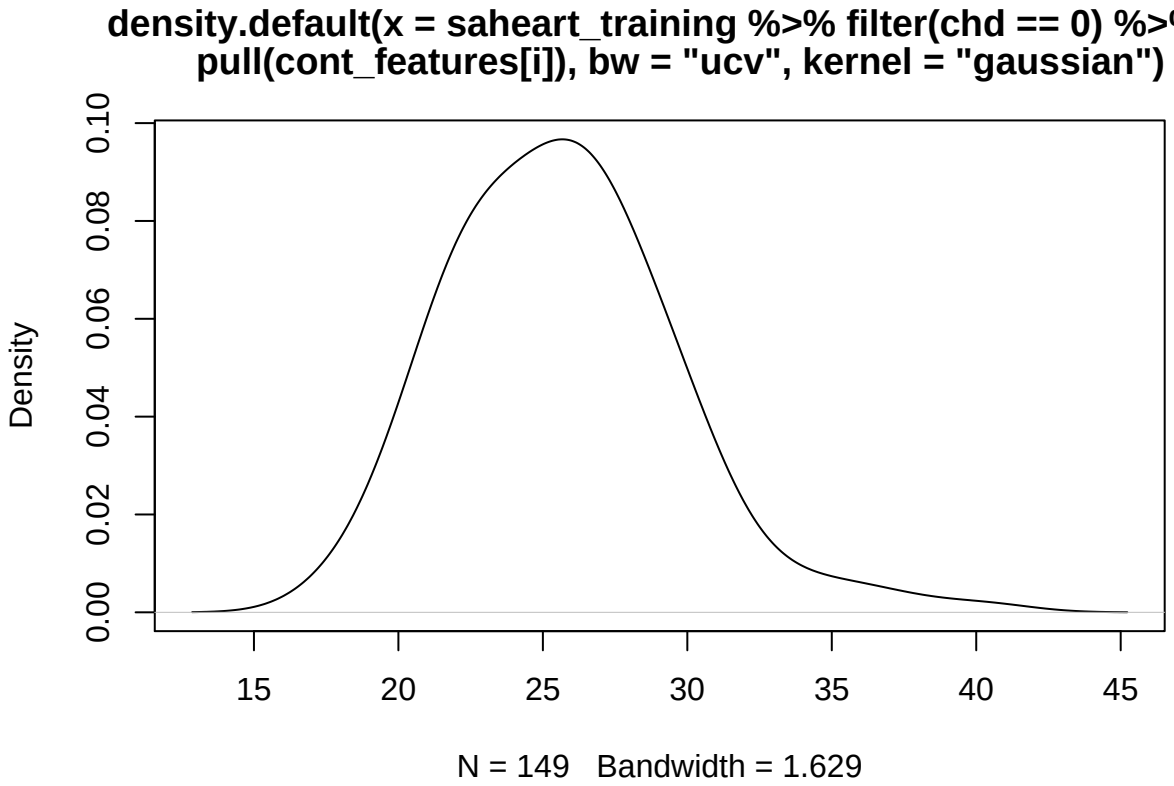

density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.01
0.02
0.03
0.04
0.05
0
10
20
30
40
50
N = 82 Bandwidth = 3.317

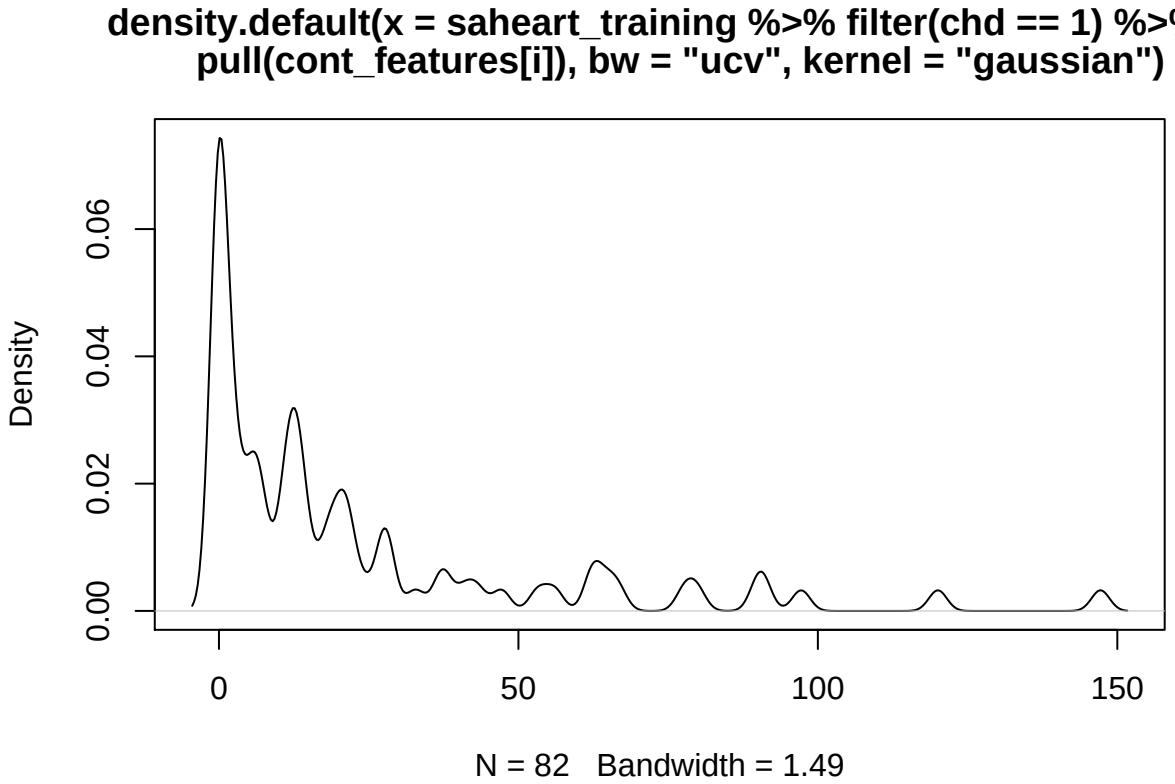

density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.01
0.02
0.03
0.04
0.05
0
10
20
30
40
50
N = 149 Bandwidth = 3.147

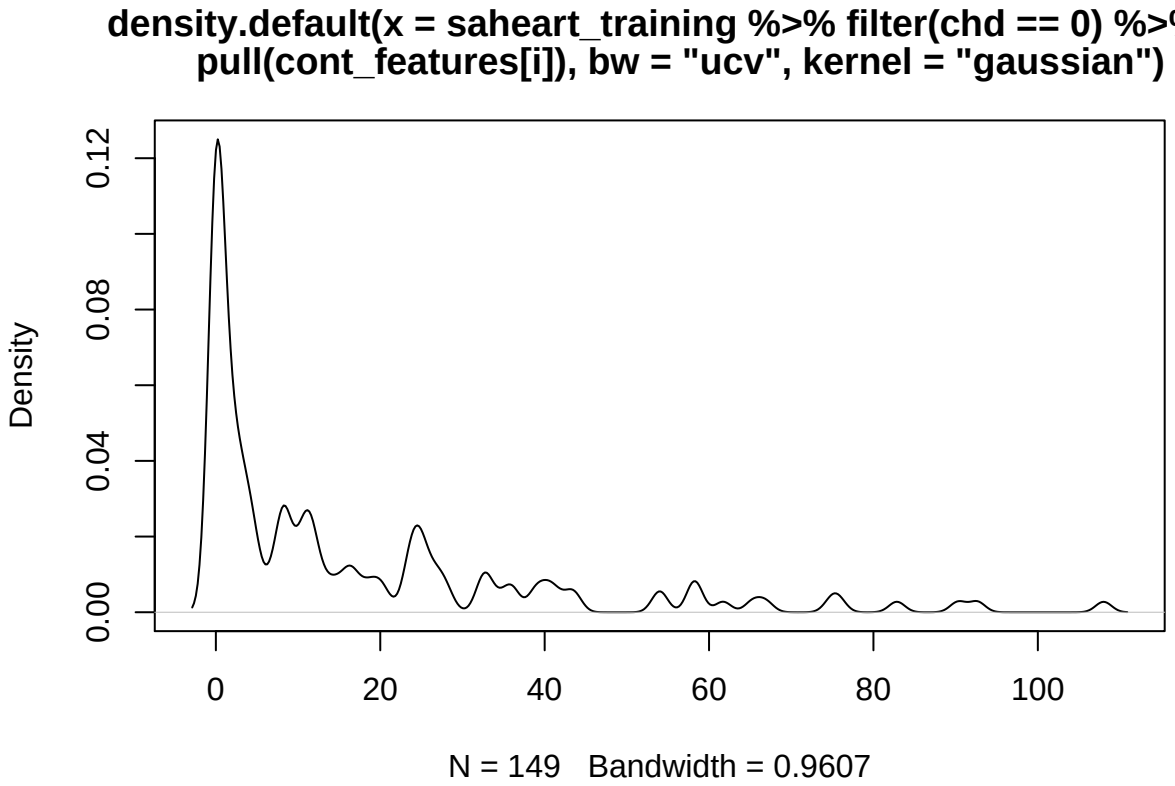

density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.000
0.010
0.020
0.030
20
30
40
50
60
70
80
90
N = 82 Bandwidth = 4.541

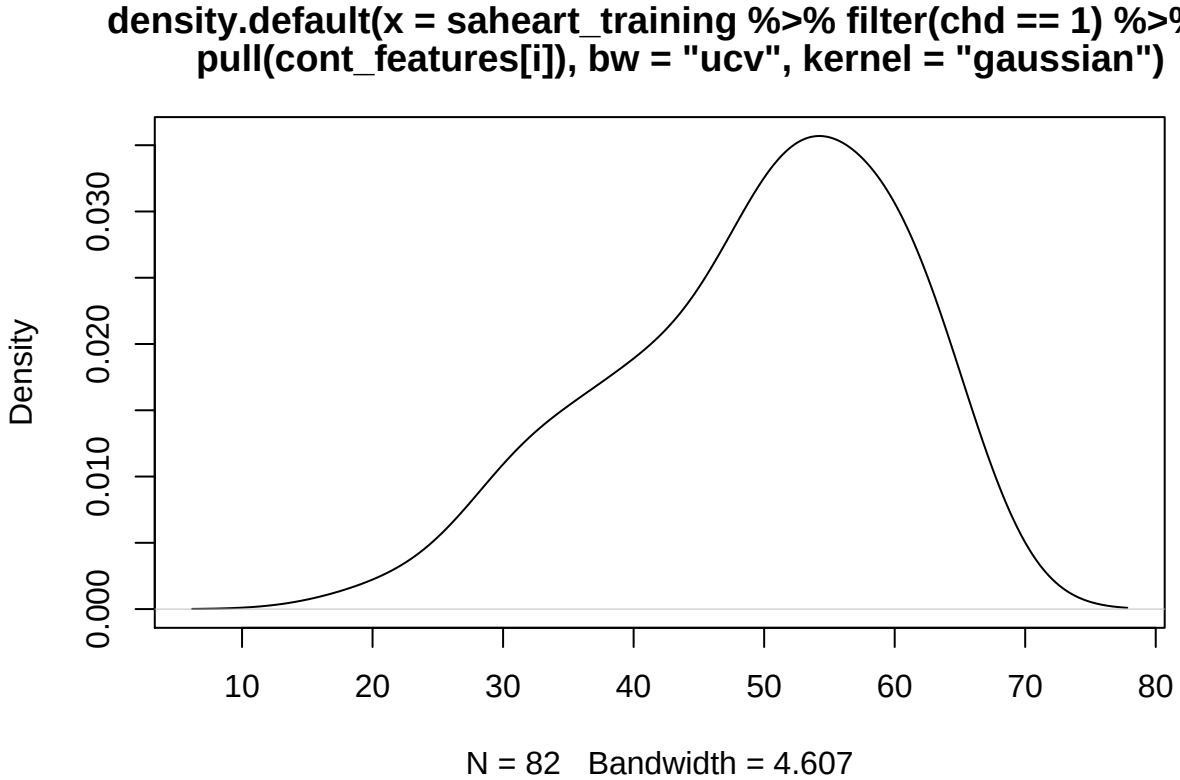

density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.01
0.02
0.03
0.04
0
20
40
60
80
N = 149 Bandwidth = 3.484

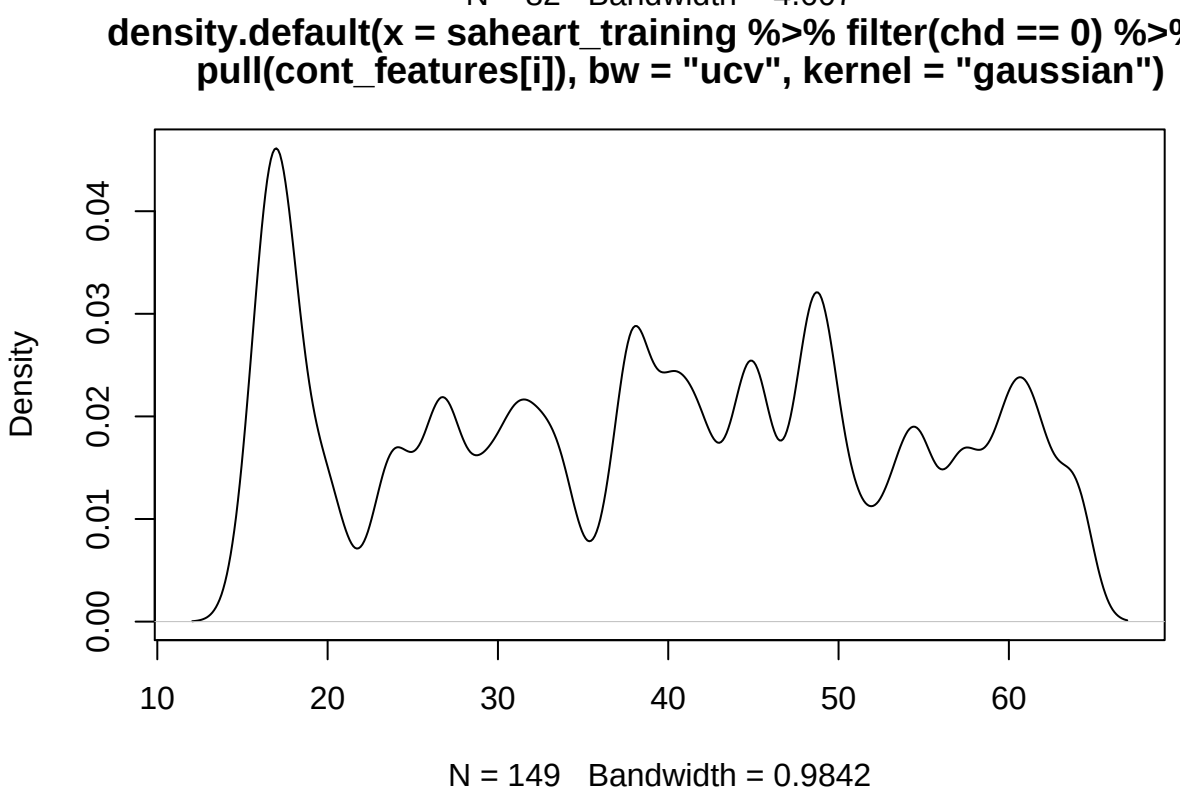

density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.02
0.04
0.06
0.08
0.10
15
20
25
30
35
40
45
N = 82 Bandwidth = 1.759


density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.02
0.04
0.06
0.08
0.10
15
20
25
30
35
40
45
N = 149 Bandwidth = 1.629


density.default(x = saheart_training %>% filter(chd == 1) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.02
0.04
0.06
0
50
100
150
N = 82 Bandwidth = 1.49


density.default(x = saheart_training %>% filter(chd == 0) %>% pull(cont_features[i]), bw = "ucv", kernel = "gaussian")
Density
0.00
0.04
0.08
0.12
0
20
40
60
80
100
N = 149 Bandwidth = 0.9607





Let's define a few helper functions that aid in the implementation of the Naive Baiyes classifier:

1. `feature_density_estimate`: Given a feature's name, value and a class `c`, returns the estimated density of the feature given `c`.

```
feature_density_estimate_helper <- function(density, feature_val) {
  step_size <- density$x[2] - density$x[1]
  index <- as.integer((feature_val - density$x[1]) / step_size)
  if(index > 1 & index <= 512) {
    return(density$y[index])
  }
  else {
    return(0)
  }
}

feature_density_estimate <- function(feature_name, feature_value, c) {
  result <- 0
  if (c == 1) {
    result <- feature_density_estimate_helper(densities.one[[which(cont_features==feature_name)]],
                                              feature_value)
  }
  else {
    result <- feature_density_estimate_helper(densities.zero[[which(cont_features==feature_name)]],
                                              feature_value)
  }
  return(result)
}
```

2. `obervation_density_estimate`: Given an observation and a class `c`, returns the estimated probability density of the observation given `c`.

```
obervation_density_estimate <- function(observation, c) {
  result <- 1
  for(feature_name in names(observation)) {
    result <- result * feature_density_estimate(feature_name, observation[feature_name], c)
  }
  return(result)
}
```

3. `naive_bayes_prob`: Given an observation and a class `c`, returns the estimated probability of the observation belonging to `c`.

```
prior_prob <- function(c) {
  return(saheart_training %>% filter(chd == c) %>% nrow() / nrow(saheart_training))
}

naive_bayes_prob <- function(observation, c) {
  likelihood <- prior_prob(c) * obervation_density_estimate(observation, c)
  support <- prior_prob(1) * obervation_density_estimate(observation, 1) +
             prior_prob(0) * obervation_density_estimate(observation, 0)
  if(support == 0) {
    return(0)
  }
  else {
    return(likelihood / support)
  }
}
```

4. `naive_bayes_predict`: Given an observation, predicts the class that it belongs to.

```
naive_bayes_predict <- function(observation) {
  prob.zero <- naive_bayes_prob(observation, 0)
  prob.one <- naive_bayes_prob(observation, 1)
  if(prob.one > prob.zero) {
    return(1)
  }
  else {
    return(0)
  }
}
```

Finally, we test our Naive Baiyes classifier on the validation data and compute its performance.

```
naive_bayes_performance <- function(observations, actual_y) {
  true_positive <- 0
  for(i in 1:nrow(observations)) {
    if (naive_bayes_predict(observations[i, ]) == actual_y[i]) {
      true_positive = true_positive + 1
    }
  }
  return(true_positive / nrow(observations))
}

result <- naive_bayes_performance(saheart_validation %>% dplyr::select_(.dots = cont_features),
                                  saheart_validation %>% pull(chd))

cat("Accuracy of Naive Bayes Classifier: ", result)
```

```
> Accuracy of Naive Bayes Classifier:  0.6493506
```

c.

```
library(MASS)
lda.fit <- lda(chd~sbp + tobacco + ldl + adiposity + typea + obesity + alcohol + age,
               data = saheart_training)
lda_predict <- predict(lda.fit,saheart_validation[,-10],type="response")
lda_predict <- as.numeric(lda_predict$class)-1
actual <- as.numeric(saheart_validation[,10]) - 1
accu <- 1-sum(abs(lda_predict - actual)) / length(actual)
cat("Accuracy of LDA: " , accu)
```

```
> Accuracy of LDA:  0.7316017
```

d.

Let's consider both the `linear` and `radial` kernels while using SVM's to perform the classifcation task at hand.

1. Support Vector Classification: Linear Kernel

```
# Import
library(e1071)

svm.linear.tune <- tune(svm, chd~., data = saheart_training, kernel = "linear",
```

```
                        ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
svm.linear.best <- svm.linear.tune$best.model
summary(svm.linear.best)
```

```
>
> Call:
> best.tune(method = svm, train.x = chd ~ ., data = saheart_training,
>     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
>     kernel = "linear")
>
>
> Parameters:
>    SVM-Type:  C-classification
>  SVM-Kernel:  linear
>        cost:  0.1
>       gamma:  0.1111111
>
> Number of Support Vectors:  148
>
>  ( 75 73 )
>
>
> Number of Classes:  2
>
> Levels:
>  0 1
```

```
svm.linear.pred <- predict(svm.linear.best, saheart_validation)
svm.linear.result <- table(svm.linear.pred, saheart_validation$chd)

cat("Accuracy of Support Vector Classifier: ", sum(diag(svm.linear.result)) / sum(svm.linear.result))
```

```
> Accuracy of Support Vector Classifier:  0.7445887
```

2. Support Vector Machine: Radial Kernel

```
svm.radial.tune <- tune(svm, chd~., data = saheart_training, kernel = "radial",
                        gamma = c(0.5, 1, 2, 3, 4),
                        ranges = list(cost = c(0.1, 1, 10, 100, 1000)))
svm.radial.best <- svm.radial.tune$best.model
summary(svm.radial.best)
```

```
>
> Call:
> best.tune(method = svm, train.x = chd ~ ., data = saheart_training,
>     ranges = list(cost = c(0.1, 1, 10, 100, 1000)), kernel = "radial",
>     gamma = c(0.5, 1, 2, 3, 4))
>
>
> Parameters:
>    SVM-Type:  C-classification
>  SVM-Kernel:  radial
>        cost:  0.1
>       gamma:  0.5 1 2 3 4
>
> Number of Support Vectors:  212
```

```
>
>  ( 82 130 )
>
>
> Number of Classes:  2
>
> Levels:
>  0 1
```

```
svm.radial.pred <- predict(svm.radial.best, saheart_validation)
svm.radial.result <- table(svm.radial.pred, saheart_validation$chd)

cat("Accuracy of Support Vector Machine: ", sum(diag(svm.radial.result)) / sum(svm.radial.result))
```

```
> Accuracy of Support Vector Machine:  0.6623377
```

The SVM with the linear kernel performs better as seen above.

e.

The below table summarizes the models and their respective accuracies on the validation set.

Model	Accuracy
Naive Bayes	0.6493506
LDA	0.7316017
SVM (Linear kernel)	0.7445887

Naive Bayes Classification is based on the assumption that the features of the given observation are indepent of each other. Looking at the low accuracy the model achieved, it is clear that there are correlated features in the dataset.

The accuracies achieved by LDA and SVM are not too far apart. A reason for their good accuracies might be because the output is in fact linearly separable. Also, LDA being a generative classifier, assumes that the samples from both the classes have the same covariance and that the density functions are normally distributed. Whereas, SVM is a discriminative classifier