Solutions to Homework 3

Nakul Camasamudram

10/18/2017

Problem 1

(a)

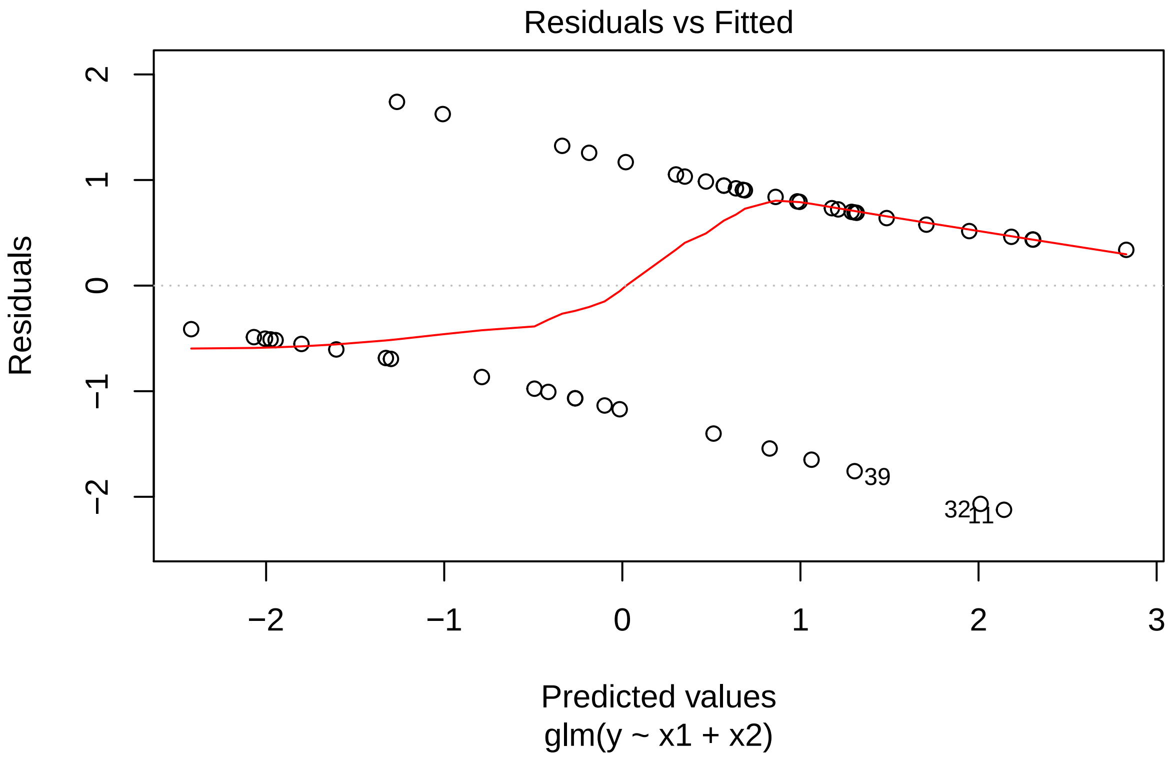
```r
set.seed(123)
prob_func_p1 <- function(x1, x2) {
  return(1 / (1 + exp(-(-3 + x1 + x2))))
}

p1_func <- function(seed_value) {
  set.seed(seed_value)
  N <- 50
  x1 <- runif(N, min=0, max=3)
  x2 <- runif(N, min=0, max=3)
  p_of_1 <- prob_func_p1(x1, x2)
  y <- rbinom(N, 1, p_of_1)
  return(data.frame(x1, x2, y))
}

training_data <- p1_func(123)
validation_data <- p1_func(456)
```

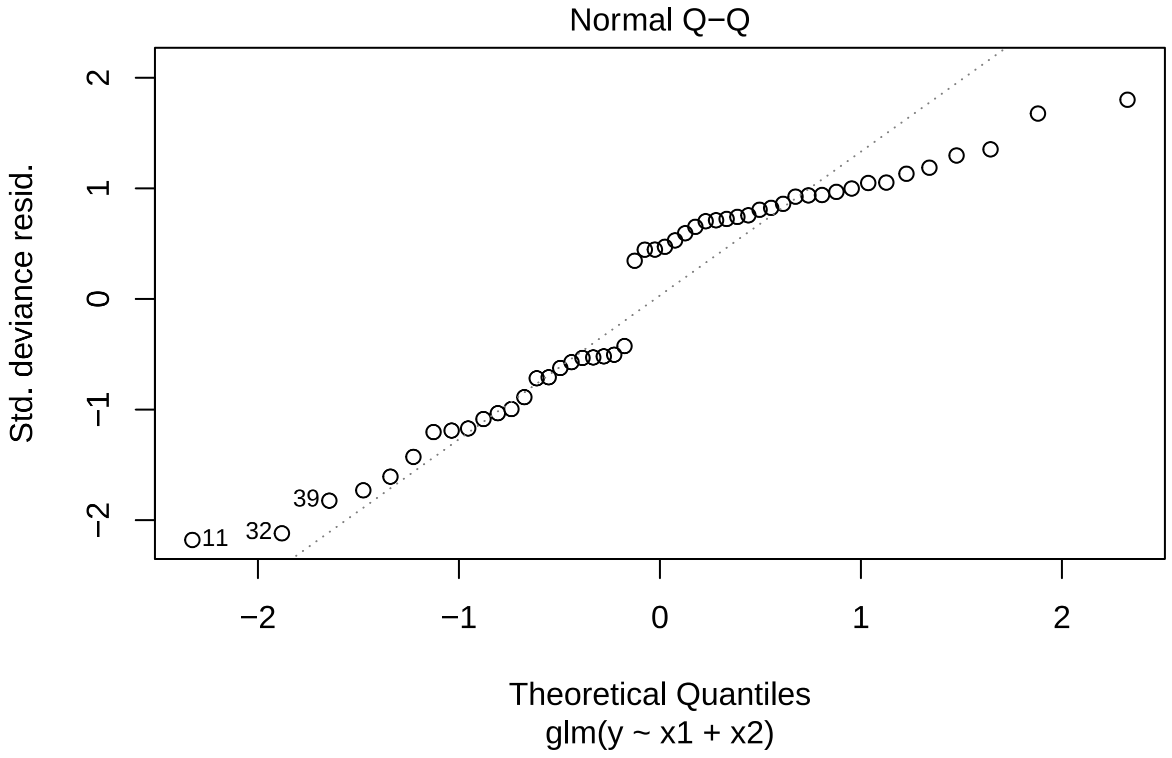
(b)

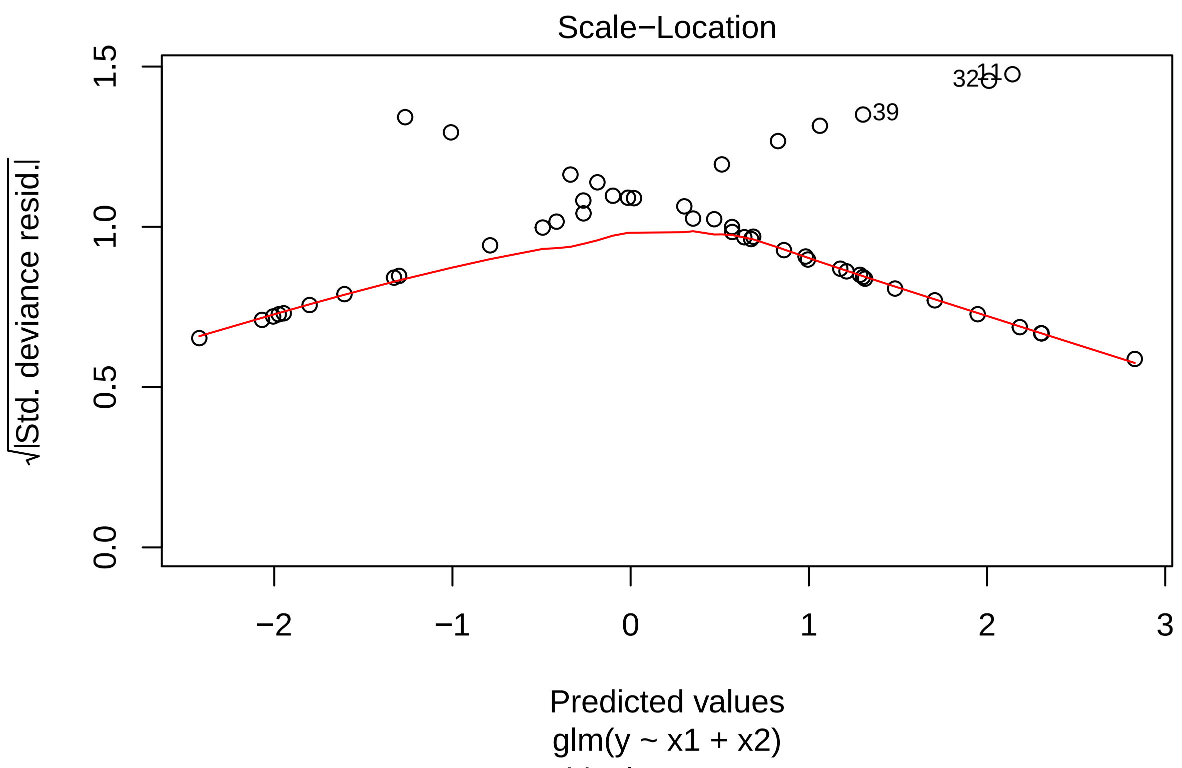
```r
# Logistic Regression
glm.fit <- glm(y ~ x1 + x2, data = training_data, family = binomial)
print(glm.fit)
```

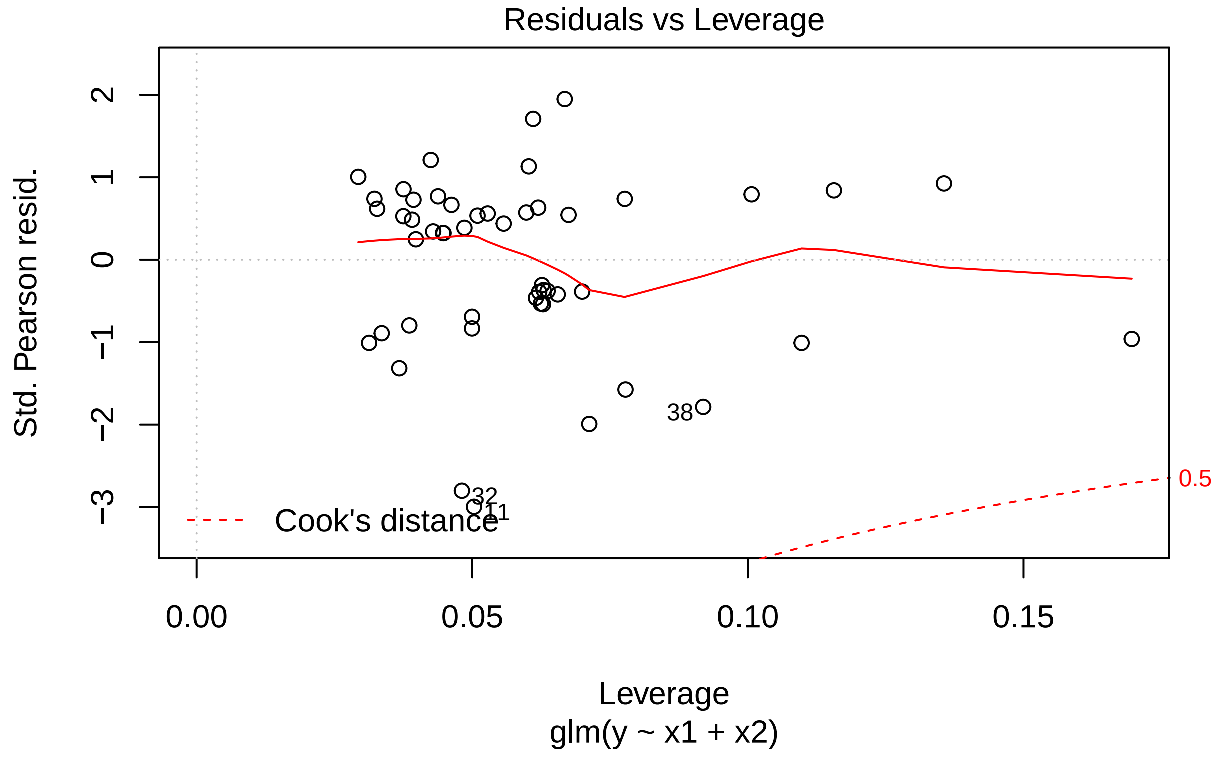
```
>
> Call:  glm(formula = y ~ x1 + x2, family = binomial, data = training_data)
>
> Coefficients:
> (Intercept)           x1           x2
>     -2.8733       0.8735       1.2573
>
> Degrees of Freedom: 49 Total (i.e. Null);  47 Residual
> Null Deviance:        68.59
> Residual Deviance: 53.38  AIC: 59.38
```

```r
plot(glm.fit)
```


Residuals vs Fitted
Residuals
Predicted values
glm(y ~ x1 + x2)
39
32
11
2
1
0
−1
−2
3


Normal Q−Q
Std. deviance resid.
Theoretical Quantiles
glm(y ~ x1 + x2)
11
32
39
2
1
0
−1
−2





```
glm.pred <- predict(glm.fit, newdata = validation_data, type = "response")
t = table(actual= validation_data$y, predict=rbinom(50, 1, glm.pred))
cat("Proportion of correctly classified observations for Logistic Regression : ", (sum(diag(t)))/sum(t)
```

```
> Proportion of correctly classified observations for Logistic Regression :  0.52
```

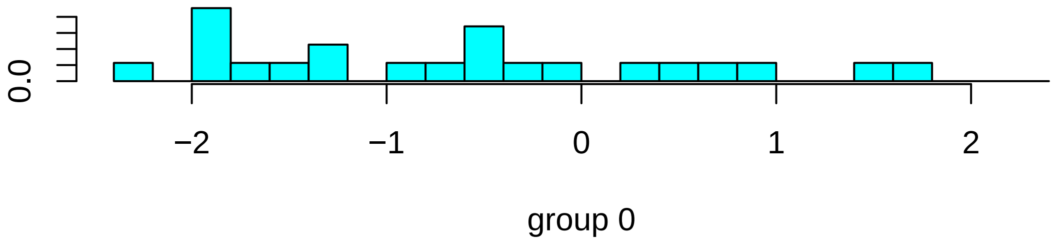
```
# LDA
library(MASS)
```

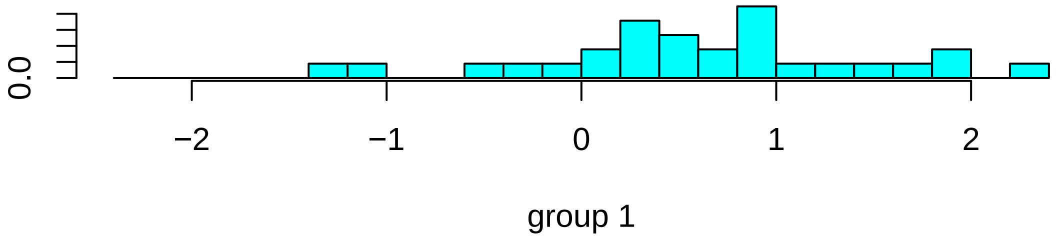
```
>
> Attaching package: 'MASS'

> The following object is masked from 'package:dplyr':
>
>     select
```

```
lda.fit <- lda(y ~ x1 + x2, data = training_data)
print(lda.fit)
```

```
> Call:
> lda(y ~ x1 + x2, data = training_data)
>
> Prior probabilities of groups:
>    0    1
> 0.44 0.56
>
> Group means:
>         x1       x2
> 0 1.242918 1.026482
> 1 1.809624 1.748980
>
> Coefficients of linear discriminants:
>          LD1
> x1 0.7276555
> x2 1.1065328
```

```
plot(lda.fit)
```



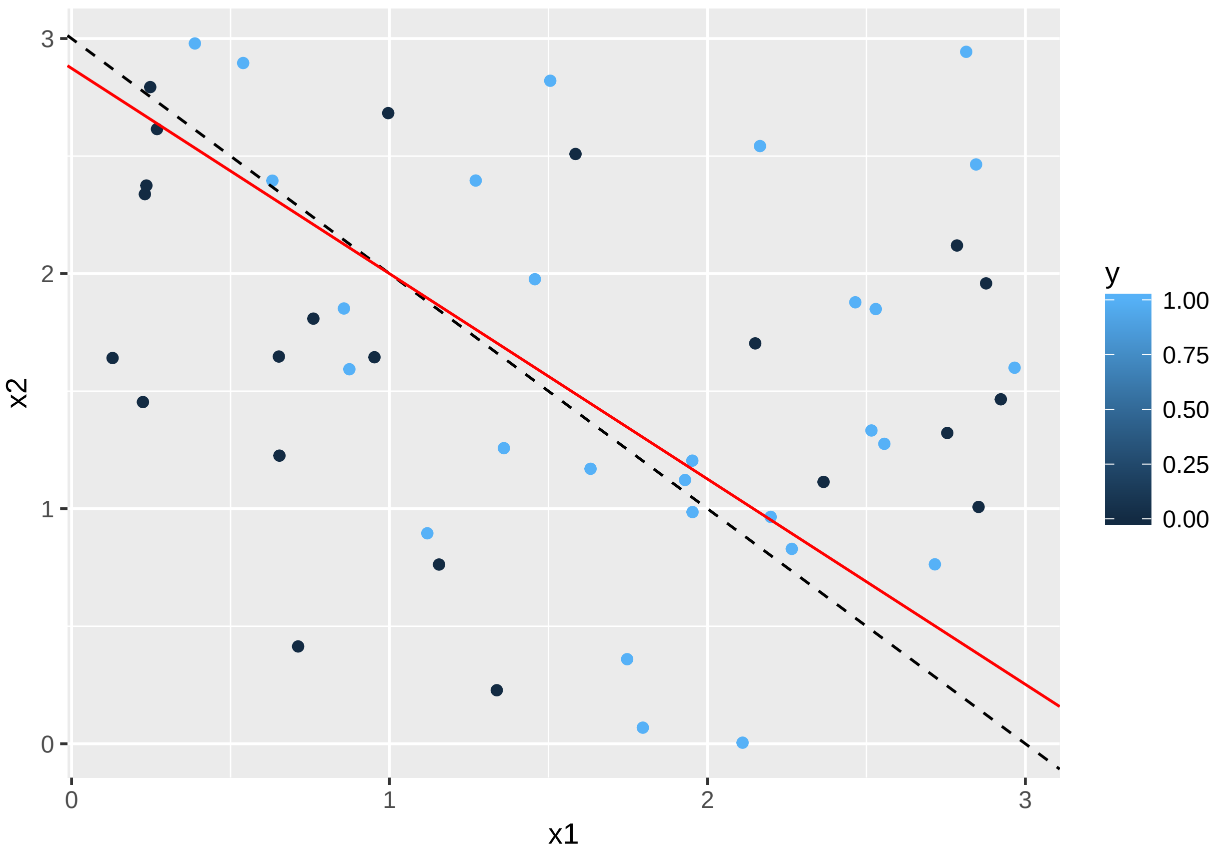


```
lda.pred <- predict(lda.fit, newdata = validation_data)
t = table(actual= validation_data$y, predict=lda.pred$class)
```

```
cat("Proportion of correctly classified observations for LDA : ", (sum(diag(t)))/sum(t))
```

```
> Proportion of correctly classified observations for LDA :  0.58
```

```
# Plots
p <- ggplot(data = validation_data, aes(x = x1, y = x2)) +
          geom_point(aes(color = y))
# True decision boundary
p <- p + geom_abline(slope = -1, intercept = 3, linetype = "dashed")
# Logistic Regression decision boundary
p <- p + geom_abline(slope = -glm.fit$coefficients[2], intercept = -glm.fit$coefficients[1], color = "r
p
```

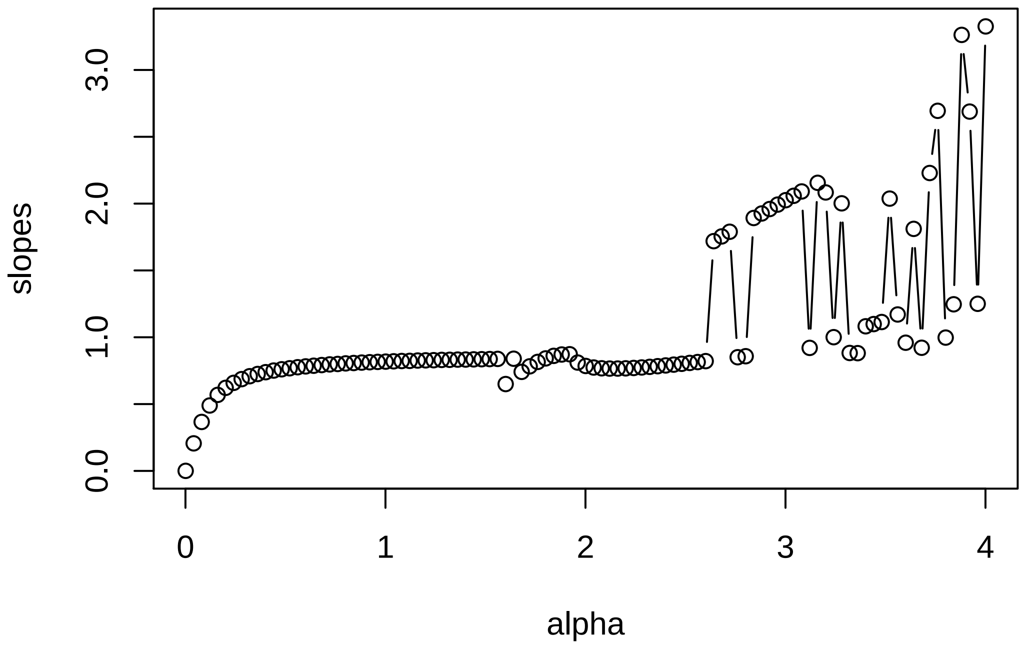


In the above graph, the dashed line is hte true decision boundary and the red line is the decision boundary for logistic regression.

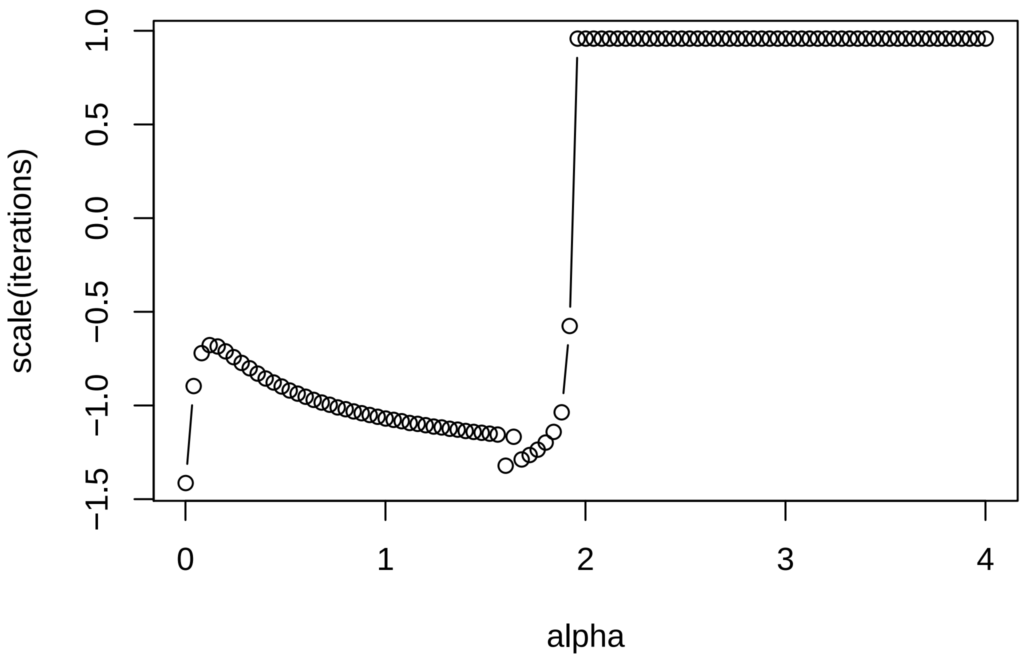
c

1. Batch gradient descent

```
batch_func <- function(learn_rate, train_df){
  delta <- 1/1000
  train_df_x <- cbind(x_0 = 1, train_df %>% dplyr::select(-y))

  b <- cbind(rep(0, (ncol(train_df_x))))
  b_prev <- cbind(rep(1,(ncol(train_df_x))))
  b_diff <- abs(b - b_prev)
  y <- train_df$y
  i <- 0
  while(i < 1000 && length(b_diff[b_diff > delta]) > 1) {
    i <- i + 1
    hyp <- 1 / (1 + exp(-1 * (as.matrix(train_df_x) %*% b)))
    b_prev <- b
    b <- b + (learn_rate / nrow(train_df)) * (t(as.matrix(train_df_x)) %*% (y-hyp))
    b_diff <- abs(b - b_prev)
  }
return(c(b, i))
}

# The below code is from Guo & Gundogdu's solutions for HW2
alpha <- seq(0.001, 4.001, 0.04)
slopes <- rep(0, length(alpha))
iterations <- rep(0, length(alpha))
for (i in 1:length(alpha))
{
  slopes[i] <- batch_func(alpha[i],training_data)[2];
  iterations[i] <- batch_func(alpha[i],training_data)[4];
}
par(mar = c(5,5,5,5))
plot(alpha, slopes, type="b")
```



```
plot(alpha, scale(iterations), type="b")
```



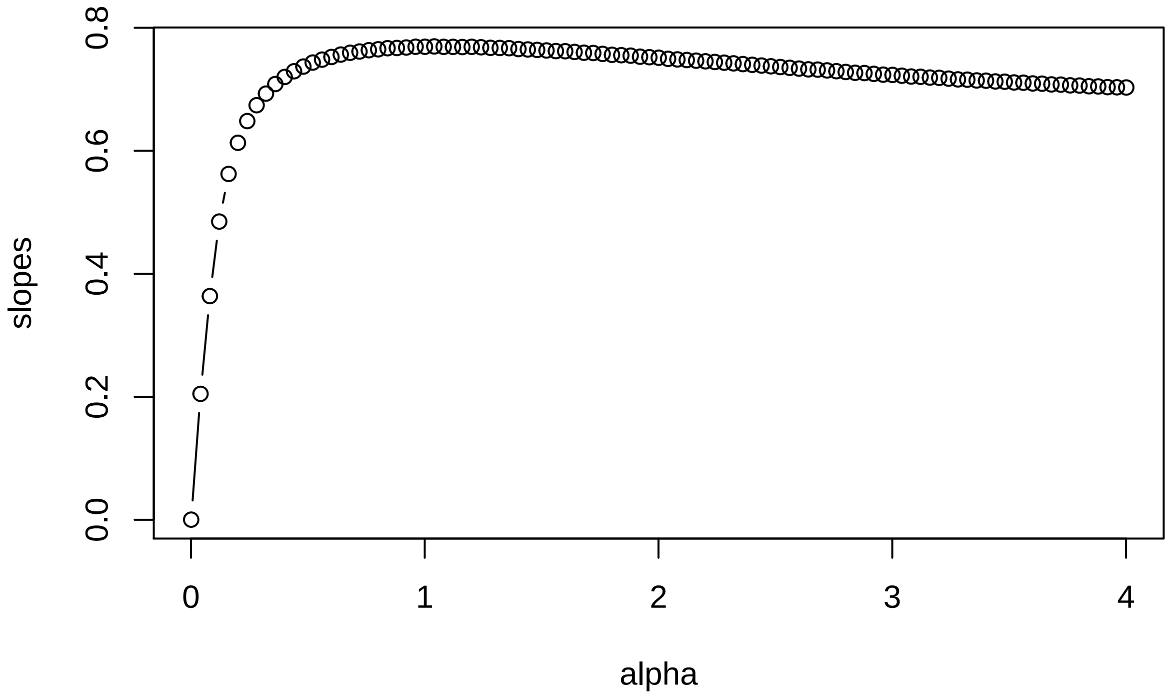
The optimal value of α seems to be 1.7.

```r
batch_learning_rate <- 1.7
```

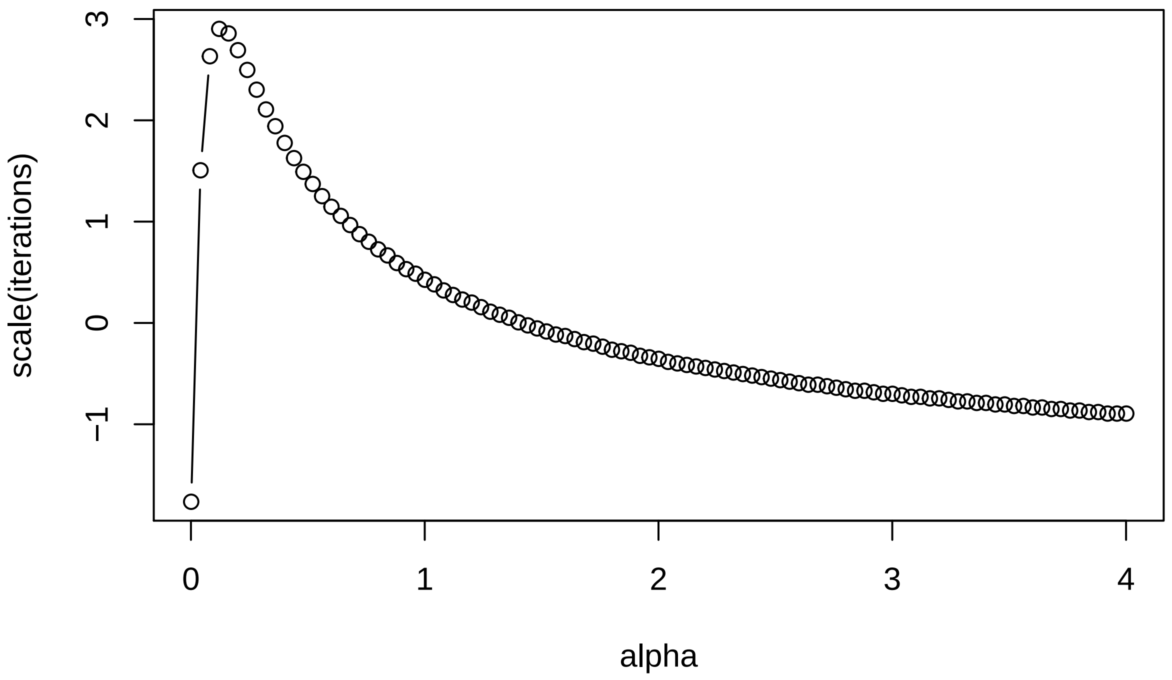
2. Stochastic gradient descent

```r
stochastic_func <- function(learn_rate, train_df){
  delta <- 1/1000
  train_df_x <- cbind(x_0 = 1, train_df %>% dplyr::select(-y))

  b <- cbind(rep(0, (ncol(train_df_x))))
  b_prev <- cbind(rep(1,(ncol(train_df_x))))
  b_diff <- abs(b - b_prev)

  y <- train_df$y

  i <- 0
  while(i < 1000 && length(b_diff[b_diff > delta]) > 1) {
    b_prev <- b
    for(j in 1:nrow(train_df_x)){
      hyp <- 1 / (1 + exp(-1 * (as.matrix(train_df_x[j,]) %*% b)))
      b <- b + (learn_rate / nrow(train_df)) * (t(as.matrix(train_df_x[j,])) %*% (y[j] - hyp))
    }
    i <- i+1
    b_diff <- abs(b - b_prev)
  }
  return(c(b,i))
}

# The below code is from Guo & Gundogdu's solutions for HW2
alpha <- seq(0.001, 4.001, 0.04)
slopes <- rep(0, length(alpha))
iterations <- rep(0, length(alpha))
for (i in 1:length(alpha))
```

```
{
  slopes[i] <- stochastic_func(alpha[i], training_data)[2];
  iterations[i] <- stochastic_func(alpha[i], training_data)[4];
}
plot(alpha,slopes,type="b")
```



```
plot(alpha,scale(iterations),type="b")
```



```
# Optimal alpha
stoch_learning_rate <- alpha[which(slopes==max(slopes))]
```

```r
# Number of iterations for the above alpha
stoch_iterations <- iterations[which(slopes==max(slopes))]

cat("Alpha: ", stoch_learning_rate)
```

```
> Alpha:  1.041
```

```r
cat(" Number of iterations: ", stoch_iterations)
```

```
>  Number of iterations:  144
```

The optimal value of α seems to be 1.041.

d

Linear Discriminant Analysis

```r
lda_func <- function(train_df, validation_df) {
  groups <- unique(train_df$y)
  mu <- data_frame()
  constant <- c()
  covar_inv <- solve(cov(train_df %>% dplyr::select(-y)))

  for(k in groups) {
    train_df_k <- train_df %>% filter(y==k) %>% dplyr::select(-y)
    mu_k <- apply(train_df_k, 2, mean)
    mu <- mu %>% bind_rows(data.frame(as.list(mu_k)))
    pi_k <-nrow(train_df_k) / nrow(train_df)
    constant_k <- (0.5 * t(mu_k) %*% covar_inv %*% matrix(mu_k)) + log(pi_k)
    constant <- c(constant, constant_k)
  }
  lda_model <- data.frame(groups = groups, constant = constant) %>% bind_cols(mu=mu)

  y_pred_func <- function(x) {
    delta_x <- -Inf
    y_pred <- NULL
    for(k in groups) {
      mu_k <- data.matrix(lda_model %>% filter(groups==k) %>% dplyr::select(-groups) %>% dplyr::select(
      delta_x_k <- x %*% covar_inv %*% t(mu_k) - (lda_model %>% filter(groups==k) %>% pull(constant))
      if(delta_x_k > delta_x) {
        y_pred <- k
        delta_x <- delta_x_k
      }
    }
    return(y_pred)
  }

  y_pred <- apply(validation_df %>% dplyr::select(-y), 1, y_pred_func)
  return(y_pred)
}

lda_prediction <- lda_func(training_data, validation_data)
t = table(actual = validation_data$y, predict = lda_prediction)
cat("Proportion of correctly classified observations for LDA : ", (sum(diag(t)))/sum(t)*100, "%")
```

```
> Proportion of correctly classified observations for LDA :  50 %
```

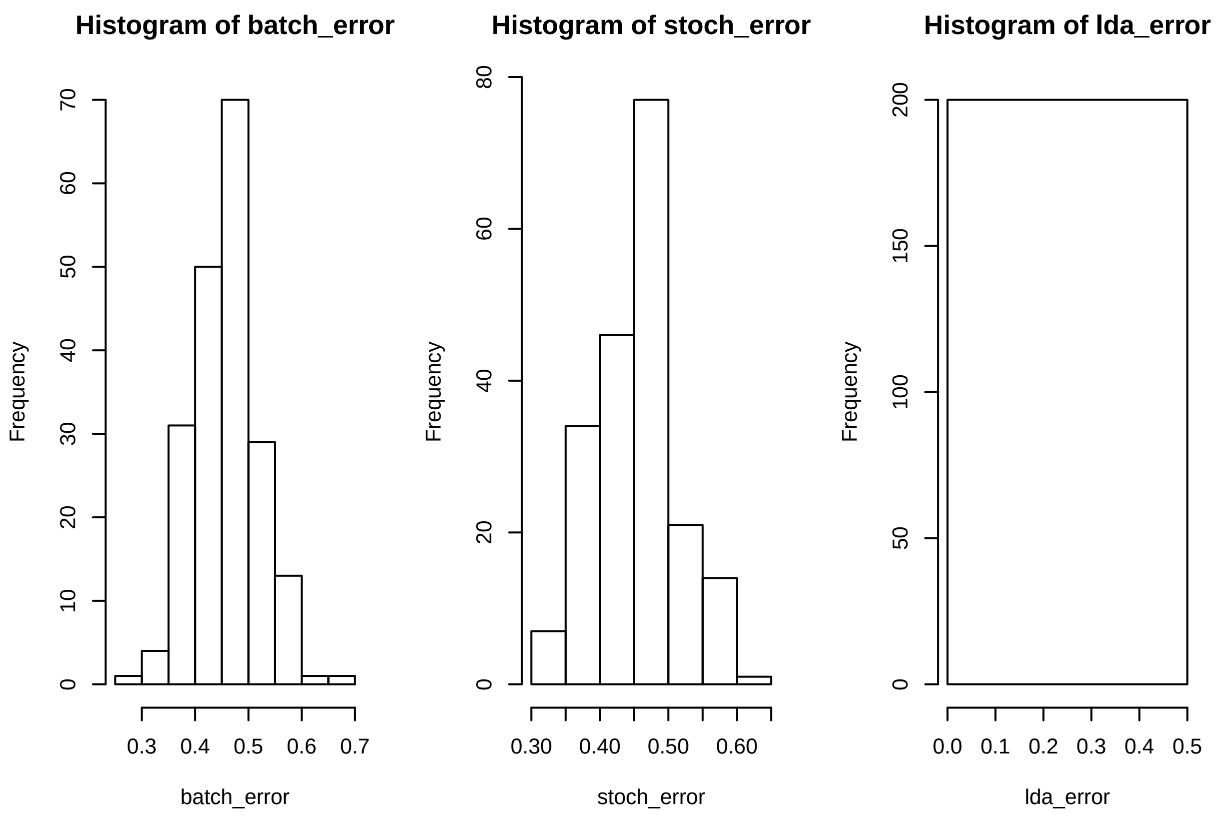
e

Functions to compute the errors for regression and LDA:

```r
# Batch and Stochastic Regression errors
reg_errors <- function(b, validation_df){
  b <- b[-1 * length(b)]
  validation_df_x <- cbind(x_0 = 1, validation_df %>% dplyr::select(-y))
  hyp <- 1 / (1 + exp(-1 * (as.matrix(validation_df_x) %*% b)))
  y_pred <- rbinom(nrow(validation_df), 1, hyp)
  t <- table(actual=validation_df$y, predict=y_pred)
  correct <- (sum(diag(t))) / sum(t)
  return(1 - correct)
}

# LDA errors
lda_errors <- function(validation_data, lda_prediction){
  t <- table(actual = validation_data$y, predict = lda_prediction)
  correct <- (sum(diag(t)))/sum(t)
  return(1-correct)
}
```

Let's find the errors and plot them

```r
# Parts of below code are from Guo & Gundogdu's HW2 solution
N <- 200

batch_error <- rep(0, N);
stoch_error <- rep(0, N);
lda_error <- rep(0, N);

for (i in 1:N)
{
  batch_error[i] <- reg_errors(batch_func(batch_learning_rate, training_data), validation_data)
  stoch_error[i] <- reg_errors(stochastic_func(stoch_learning_rate, training_data), validation_data)
  lda_error[i] <- lda_errors(validation_data, lda_func(training_data, validation_data))
}

par(mfrow=c(1, 3))
hist(batch_error)
hist(stoch_error)
hist(lda_error)
```



The error distributions of batch and stochastic gradient descents look similar and sort of approximate the normal distribution. The LDA error is constant at 0.5 at every iteration.

Problem 2

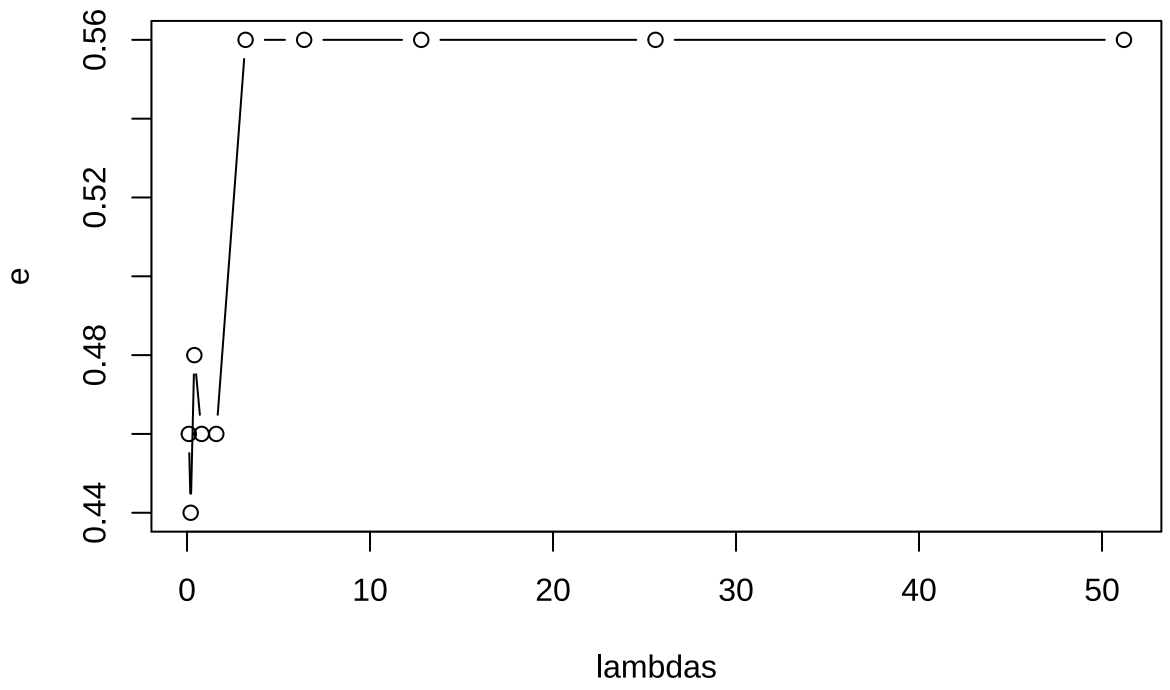
a

```r
prob_p2_func <- function(x) {
  return(1/(1 + exp(-(-3 + x))))
}

p2_func <- function(seed_value) {
  set.seed(seed_value)
  N <- 50
  x1 <- runif(N, min=0, max=3)
  x2 <- runif(N, min=0, max=3)
  x3 <- 0.8 * x2 + rnorm(N, mean = 0, sd = sqrt(0.75))
  x4 <- 0.8 * x2 + rnorm(N, mean = 0, sd = sqrt(0.75))
  x5 <- runif(N, min=0, max=3)
  x6 <- runif(N, min=0, max=3)
  x7 <- runif(N, min=0, max=3)
  x8 <- runif(N, min=0, max=3)
  x9 <- runif(N, min=0, max=3)
  x10 <- runif(N, min=0, max=3)
  x11 <- runif(N, min=0, max=3)
  x12 <- runif(N, min=0, max=3)
  p_of_1 <- prob_p2_func(x1 + x2)
  y <- rbinom(N, 1, p_of_1)
  return(data.frame(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, y))
}

training_data <- p2_func(123)
validation_data <- p2_func(456)
```

b

1. Batch gradient descent with regularization

```
reg_batch_func <- function(learn_rate, train_df, lambda){
  delta <- 1/1000
  train_df_x <- cbind(x_0 = 1, train_df %>% dplyr::select(-y))

  b <- cbind(rep(0, (ncol(train_df_x))))
  b_prev <- cbind(rep(1,(ncol(train_df_x))))
  b_diff <- abs(b - b_prev)
  y <- train_df$y

  i <- 0
  while(i < 1000 && length(b_diff[b_diff > delta]) > 1) {
    i <- i + 1
    cost <- 1 / (1 + exp(-1 * (as.matrix(train_df_x) %*% b)))
    b_prev <- b
    b <- b + (learn_rate / nrow(train_df)) * (t(as.matrix(train_df_x)) %*% (y - cost) - (b * lambda))
    b_diff <- abs(b - b_prev)
  }
return(c(b, i))
}


# I chose values of lambda by doubling them each time.
# I saw this technique in one of Andrew Ng's Machine Learning videos
lambdas <- c()
i = 0.1
while(length(lambdas) < 10){
  lambdas <- c(lambdas, i)
  i <- i*2
}

# Obtained from previous questions
e <- rep(0, length(lambdas))

for(i in 1:length(lambdas)){
  e[i] <- reg_errors(reg_batch_func(batch_learning_rate, training_data, lambdas[i]), validation_data)
}

plot(lambdas, e, type="b")
```



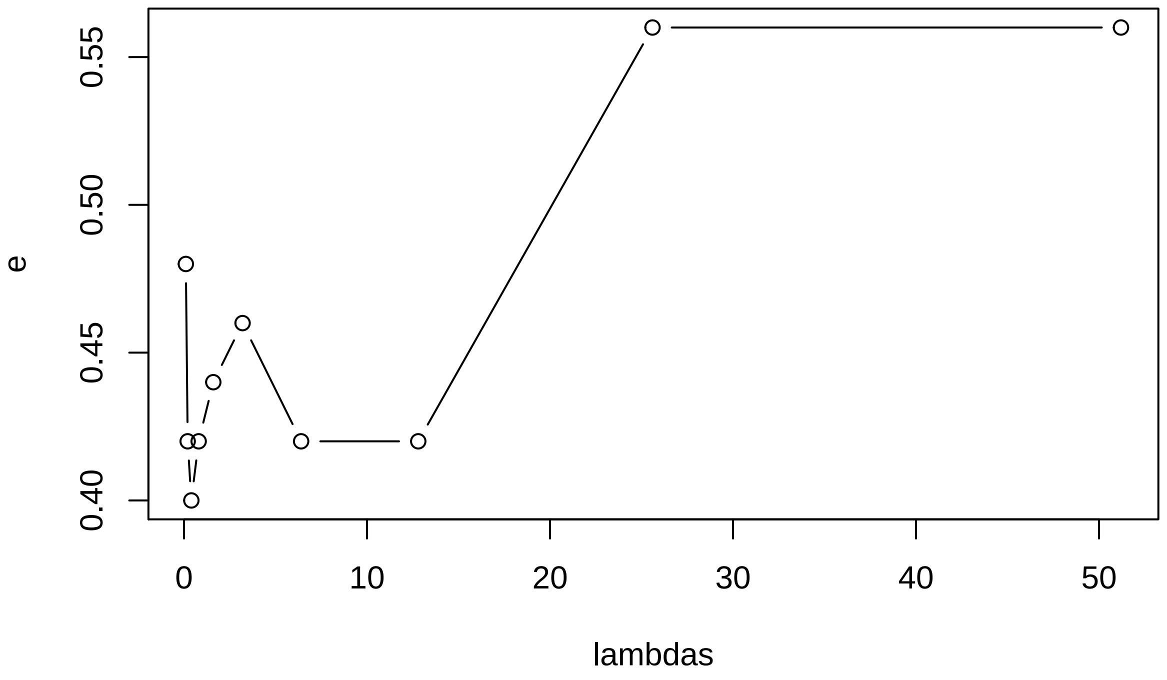
```r
# Optimum lambda
batch_opt_lambda <- lambdas[which(e==min(e))][1]
batch_opt_lambda
```

```
> [1] 0.2
```

2. Stochastic gradient descent

```r
reg_stochastic_func <- function(learn_rate, train_df, lambda){
  delta <- 1/1000
  train_df_x <- cbind(x_0 = 1, train_df %>% dplyr::select(-y))

  b <- cbind(rep(0, (ncol(train_df_x))))
  b_prev <- cbind(rep(1,(ncol(train_df_x))))
  b_diff <- abs(b - b_prev)

  y <- train_df$y

  i <- 0
  while(i < 1000 && length(b_diff[b_diff > delta]) > 1) {
    b_prev <- b
    for(j in 1:nrow(train_df_x)){
      cost <- 1 / (1 + exp(-1 * (as.matrix(train_df_x[j,]) %*% b)))
      b <- b + (learn_rate / nrow(train_df)) * (t(as.matrix(train_df_x[j,])) %*% (y[j] - cost) - (b * la
    }
    i <- i+1
    b_diff <- abs(b - b_prev)
  }
  return(c(b,i))
}

# I chose values of lambda by doubling them each time.
# I saw this technique in one of Andrew Ng's Machine Learning videos
```

```
lambdas = c()
i = 0.1
while(length(lambdas) < 10){
  lambdas <- c(lambdas, i)
  i <- i*2
}

# Obtained from previous questions
e <- rep(0, length(lambdas))

for(i in 1:length(lambdas)){
  e[i] <- reg_errors(reg_batch_func(stoch_learning_rate, training_data, lambdas[i]), validation_data)
}

plot(lambdas, e, type="b")
```

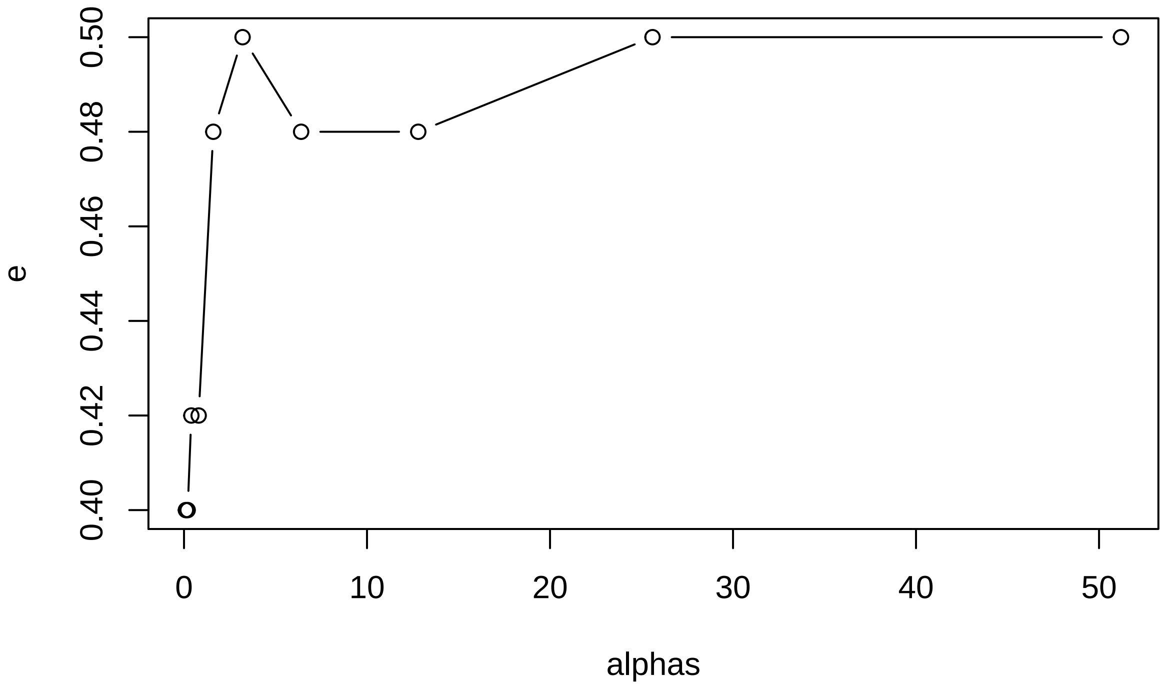


```
# Optimum lambda
stochastic_opt_lambda <- lambdas[which(e==min(e))][1]
stochastic_opt_lambda
```

```
> [1] 0.4
```

c

```
lda_reg_func <- function(train_df, validation_df, alpha) {
  groups <- unique(train_df$y)
  mu <- data_frame()
  constant <- c()
  covar_inv <- (alpha*cov(train_df %>% dplyr::select(-y)) + (1 - alpha)*(mean(diag(cov(train_df %>% dply

  for(k in groups) {
    train_df_k <- train_df %>% filter(y==k) %>% dplyr::select(-y)

    mu_k <- apply(train_df_k, 2, mean)
    mu <- mu %>% bind_rows(data.frame(as.list(mu_k)))
    pi_k <-nrow(train_df_k) / nrow(train_df)

    constant_k <- (0.5 * t(mu_k) %*% covar_inv %*% matrix(mu_k)) + log(pi_k)
    constant <- c(constant, constant_k)
  }
  lda_reg_model <- data.frame(groups = groups, constant = constant) %>% bind_cols(mu=mu)

  y_pred_func <- function(x) {
    delta_x <- -Inf
    y_pred <- NULL
    for(k in groups) {
      mu_k <- data.matrix(lda_reg_model %>% filter(groups==k) %>% dplyr::select(-groups) %>% dplyr::sel
      delta_x_k <- x %*% covar_inv %*% t(mu_k) - (lda_reg_model %>% filter(groups==k) %>% pull(constant
      if(delta_x_k > delta_x) {
        y_pred <- k
        delta_x <- delta_x_k
      }
    }
    return(y_pred)
  }

  y_pred <- apply(validation_df %>% dplyr::select(-y), 1, y_pred_func)
  return(y_pred)
}


# I chose values of alphas by doubling them each time.
# I saw this technique in one of Andrew Ng's Machine Learning videos
alphas <- c()
i = 0.1
while(length(alphas) < 10){
  alphas <- c(alphas, i)
  i <- i*2
}

# Obtained from previous questions
e <- rep(0, length(alphas))

for(i in 1:length(alphas)){
  e[i] <- lda_errors(validation_data, lda_reg_func(training_data, validation_data, alphas[i]))
}

plot(alphas, e, type="b")
```

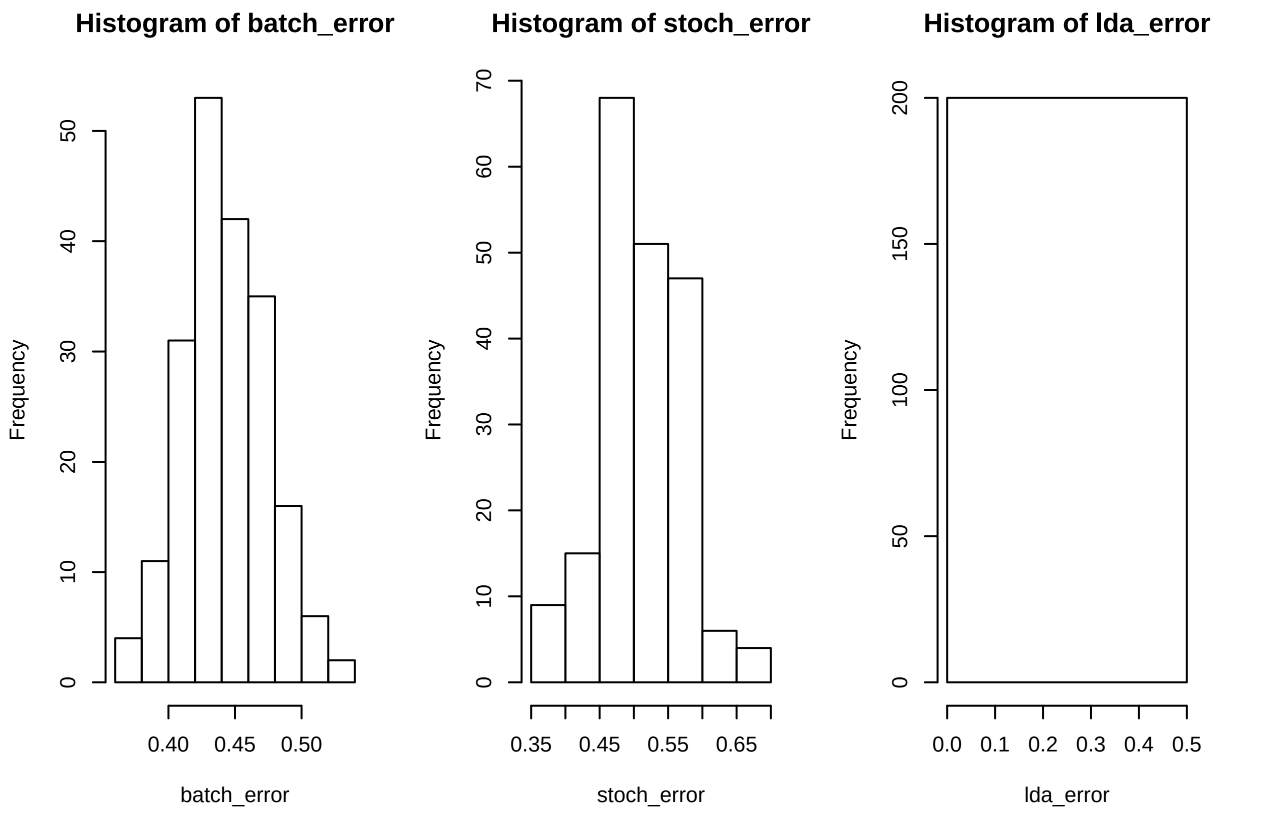


```
# Optimum alpha

lda_opt_alpha <- alphas[which(e==min(e))][1]
lda_opt_alpha
```

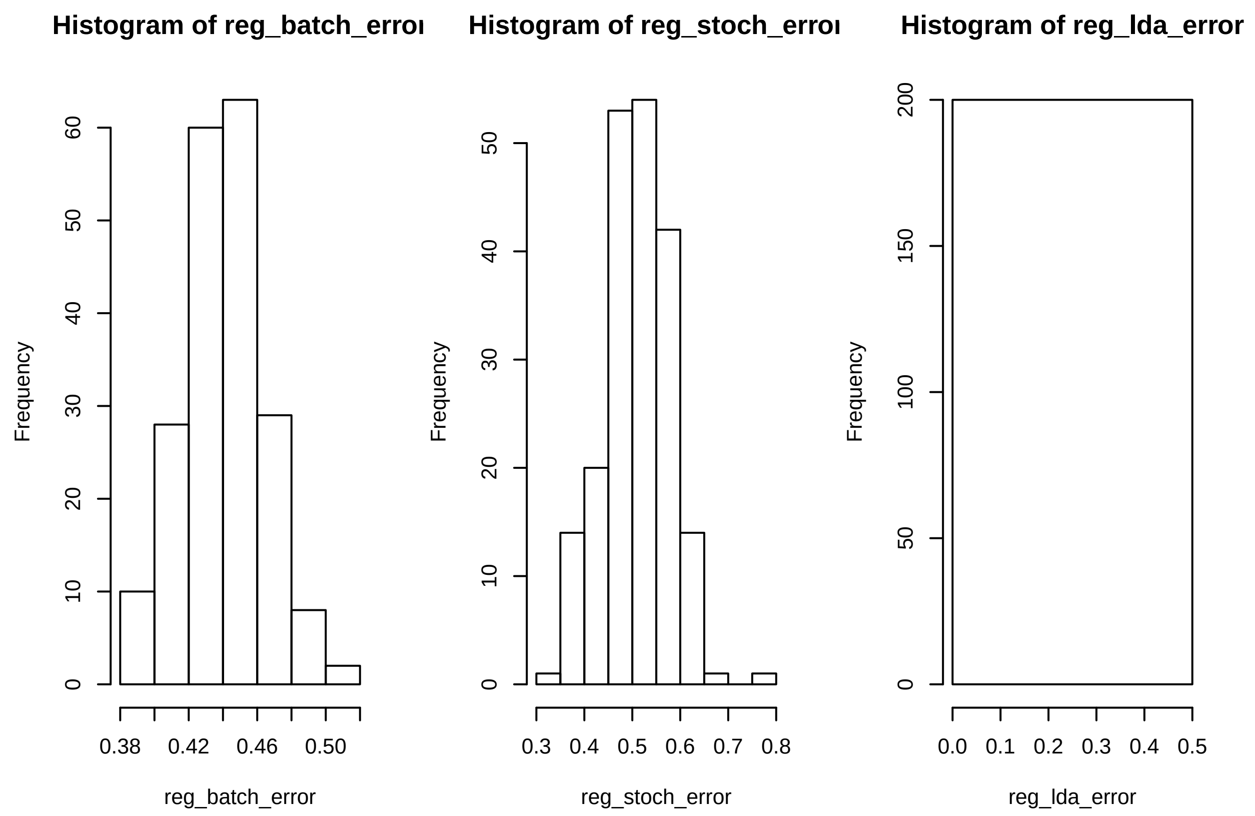
```
> [1] 0.1
```

d

```
# Parts of below code are from Guo & Gundogdu's HW2 solution
N <- 200

batch_error <- rep(0, N);
stoch_error <- rep(0, N);
lda_error <- rep(0, N);
reg_batch_error <- rep(0, N);
reg_stoch_error <- rep(0, N);
reg_lda_error <- rep(0, N);

for (i in 1:N)
{
  batch_error[i] <- reg_errors(batch_func(batch_learning_rate, training_data), validation_data)
  stoch_error[i] <- reg_errors(stochastic_func(stoch_learning_rate, training_data), validation_data)
  lda_error[i] <- lda_errors(validation_data, lda_func(training_data, validation_data))

  reg_batch_error[i] <- reg_errors(reg_batch_func(batch_learning_rate, training_data, batch_opt_lambda)
  reg_stoch_error[i] <- reg_errors(reg_stochastic_func(stoch_learning_rate, training_data, stochastic_op
  reg_lda_error[i] <- lda_errors(validation_data, lda_reg_func(training_data, validation_data, lda_opt_a
}

par(mfrow = c(1, 3))
hist(batch_error)
hist(stoch_error)
hist(lda_error)
```



```
hist(reg_batch_error)
hist(reg_stoch_error)
hist(reg_lda_error)
```



e

```r
# Mean and variance of errors

cat("Batch gradient descent. Mean: ", mean(batch_error), " Variance: ", var(batch_error), " Median: ",
```

```
> Batch gradient descent. Mean:  0.4527  Variance:  0.001037899  Median:  0.46
```

```r
cat("Stochastic gradient descent. Mean: ", mean(stoch_error), " Variance: ", var(stoch_error), " Median
```

```
> Stochastic gradient descent. Mean:  0.5178  Variance:  0.003552925  Median:  0.52
```

```r
cat("LDA Mean: ", mean(lda_error), " Variance: ", var(lda_error), " Median: ", median(lda_error), "\n")
```

```
> LDA Mean:  0.48  Variance:  0  Median:  0.48
```

```r
cat("Regularized Batch gradient descent. Mean: ", mean(reg_batch_error), " Variance: ", var(reg_batch_e
```

```
> Regularized Batch gradient descent. Mean:  0.4502  Variance:  0.0006311156  Median:  0.46
```

```r
cat("Regularized Stochastic gradient descent. Mean: ", mean(reg_stoch_error), " Variance: ", var(reg_st
```

```
> Regularized Stochastic gradient descent. Mean:  0.5157  Variance:  0.004642724  Median:  0.52
```

```r
cat("Regularized LDA. Mean: ", mean(reg_lda_error), " Variance: ", var(reg_lda_error), " Median: ", med
```

```
> Regularized LDA. Mean:  0.4  Variance:  0  Median:  0.4
```

The average error of batch gradient descent is better than stochastic. Regularization seems to have helped reduce the average error albeit not by much. Similarly, regularization seems to have reduced the error for LDA.

Problem 3

a Given an input instance x with two predictors $X1$ and $X2$ such as $x_i = [1 \;\; x_i^{(1)} \;\; x_i^{(2)}]$. The group associated with it is as below, considering that there are three output classes.

$$\hat{f}(x) = max(\begin{bmatrix} P(y=1|x) \\ P(y=2|x) \\ P(y=3|x) \end{bmatrix}) = \frac{1}{\sum_{k=1}^{K} e^{(\beta_k)^T x}} \cdot \begin{bmatrix} e^{(\beta_1)^T x} \\ e^{(\beta_2)^T x} \\ e^{(\beta_3)^T x} \end{bmatrix}$$

Here, a $\beta_i = \left[\beta_i^{(0)} \beta_i^{(1)} \beta_i^{(2)}\right]$. Hence, the number of parameters is 3.

b Since X_1 is categorical, dummy variables $X_{11}, X_{12}, X_{13}, X_{14}$ can be used to represent the classes of X_1. Then, given an input instance $x_i = [1 \;\; x_i^{(11)} \;\; x_i^{(12)} \;\; x_i^{(13)} \;\; x_i^{(14)} \;\; x_i^{(2)}]$, the group associated with it is computed as in $3a$..

However, $\beta_i = \left[\beta_i^{(0)} \beta_i^{(11)} \beta_i^{(12)} \beta_i^{(13)} \beta_i^{(14)} \beta_i^{(2)}\right]$. Hence, the number of parameters is 6.

Problem 4

a. Data exploration

```r
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
```

```r
head(saheart)
```

```
>   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
> 1 160   12.00 5.73     23.11 Present    49   25.30   97.20  52   1
> 2 144    0.01 4.41     28.61  Absent    55   28.87    2.06  63   1
> 3 118    0.08 3.48     32.28 Present    52   29.14    3.81  46   0
> 4 170    7.50 6.41     38.03 Present    51   31.99   24.26  58   1
> 5 134   13.60 3.50     27.78 Present    60   25.99   57.34  49   1
> 6 132    6.20 6.47     36.21 Present    62   30.77   14.14  45   0
```

```r
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.

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

```
>   sbp tobacco  ldl adiposity famhist typea obesity alcohol age chd
> 1 160   12.00 5.73     23.11       1    49   25.30   97.20  52   1
> 2 144    0.01 4.41     28.61       0    55   28.87    2.06  63   1
> 3 118    0.08 3.48     32.28       1    52   29.14    3.81  46   0
> 4 170    7.50 6.41     38.03       1     51    31.99    24.26  58    1
> 5 134   13.60 3.50     27.78       1     60    25.99    57.34  49    1
> 6 132    6.20 6.47     36.21       1     62    30.77    14.14  45    0
```

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

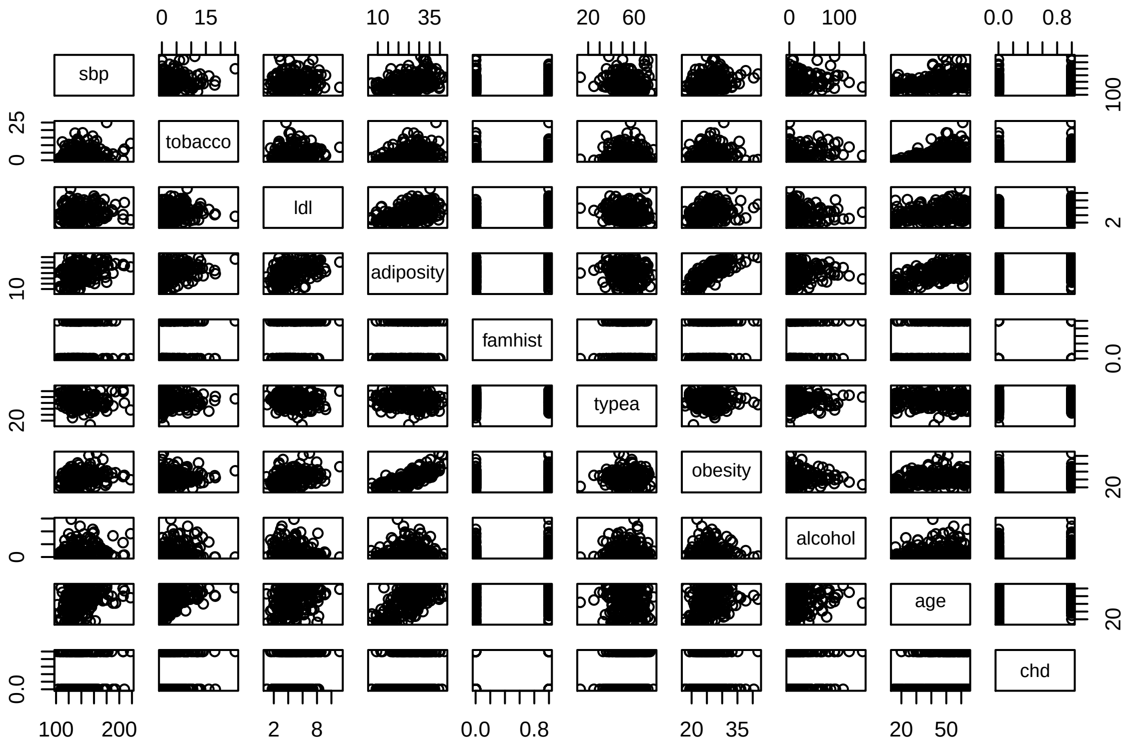
One variable summary statistics

```r
summary(saheart_training)
```

```
>       sbp           tobacco            ldl           adiposity
>  Min.   :102.0   Min.   : 0.000   Min.   : 0.980   Min.   : 6.74
>  1st Qu.:126.0   1st Qu.: 0.000   1st Qu.: 3.230   1st Qu.:20.06
>  Median :134.0   Median : 2.000   Median : 4.330   Median :25.78
>  Mean   :138.6   Mean   : 3.461   Mean   : 4.575   Mean   :25.21
>  3rd Qu.:148.0   3rd Qu.: 5.450   3rd Qu.: 5.595   3rd Qu.:30.41
>  Max.   :218.0   Max.   :25.010   Max.   :11.170   Max.   :42.17
>     famhist           typea          obesity         alcohol
>  Min.   :0.0000   Min.   :13.00   Min.   :17.75   Min.   :  0.00
>  1st Qu.:0.0000   1st Qu.:47.00   1st Qu.:23.20   1st Qu.:  0.51
>  Median :0.0000   Median :53.00   Median :25.91   Median :  8.42
>  Mean   :0.4286   Mean   :53.47   Mean   :25.92   Mean   : 18.52
>  3rd Qu.:1.0000   3rd Qu.:60.00   3rd Qu.:27.97   3rd Qu.: 24.95
>  Max.   :1.0000   Max.   :77.00   Max.   :41.76   Max.   :147.19
>       age             chd
>  Min.   :15.00   Min.   :0.000
>  1st Qu.:31.00   1st Qu.:0.000
>  Median :45.00   Median :0.000
>  Mean   :42.51   Mean   :0.355
>  3rd Qu.:55.00   3rd Qu.:1.000
>  Max.   :64.00   Max.   :1.000
```

Two variable summary statistics

```r
pairs(saheart_training) # scatterplot matrix
```

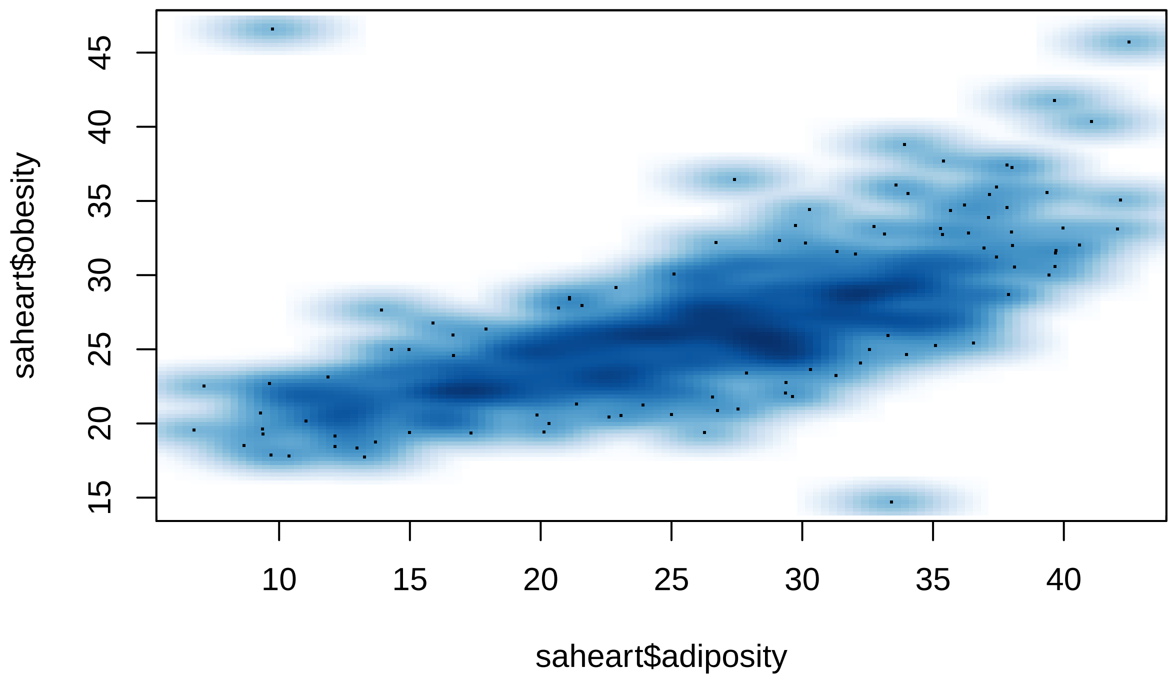


```r
round(cor(saheart_training), digits = 2)
```

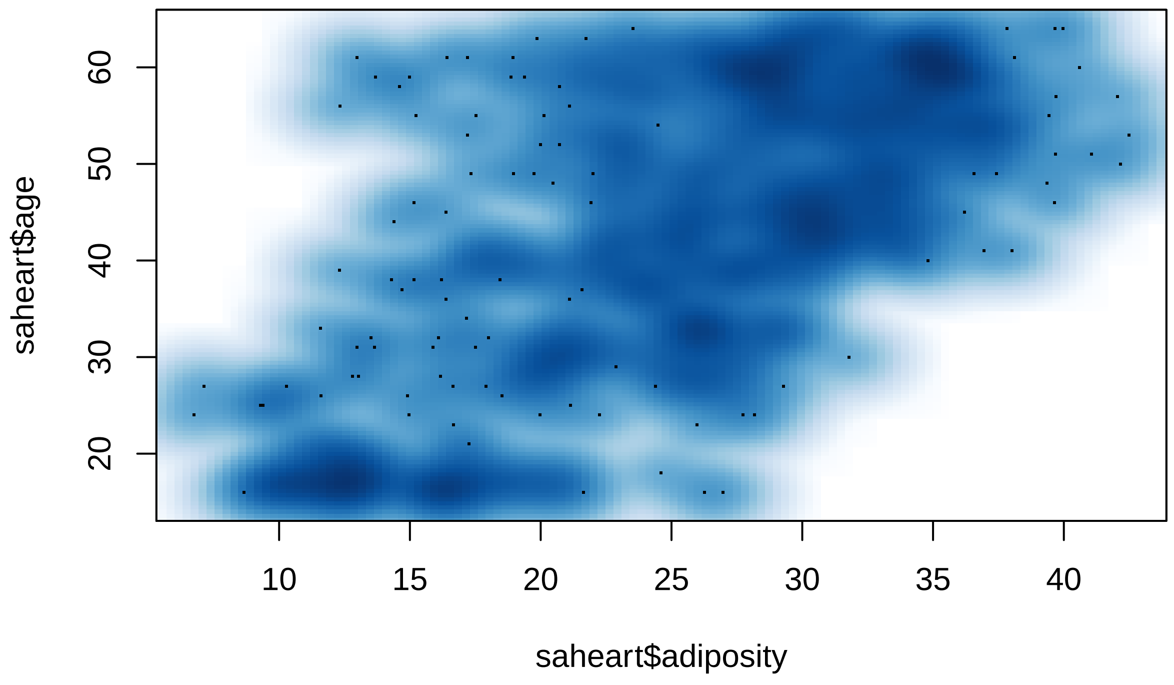
```
>              sbp tobacco   ldl adiposity famhist typea obesity alcohol
> sbp         1.00    0.17  0.14      0.34    0.11 -0.07    0.25    0.18
> tobacco     0.17    1.00  0.17      0.29    0.17 -0.01    0.10    0.23
> ldl         0.14    0.17  1.00      0.43    0.21 -0.05    0.34   -0.09
> adiposity   0.34    0.29  0.43      1.00    0.17 -0.12    0.77    0.08
> famhist     0.11    0.17  0.21      0.17    1.00 -0.01    0.11    0.08
> typea      -0.07   -0.01 -0.05     -0.12   -0.01  1.00    0.00    0.02
> obesity     0.25    0.10  0.34      0.77    0.11  0.00    1.00    0.01
> alcohol     0.18    0.23 -0.09      0.08    0.08  0.02    0.01    1.00
> age         0.38    0.46  0.33      0.67    0.28 -0.12    0.32    0.17
> chd         0.17    0.29  0.24      0.23    0.33  0.12    0.10    0.12
>              age  chd
> sbp         0.38 0.17
> tobacco     0.46 0.29
> ldl         0.33 0.24
> adiposity   0.67 0.23
> famhist     0.28 0.33
> typea      -0.12 0.12
> obesity     0.32 0.10
> alcohol     0.17 0.12
> age         1.00 0.35
> chd         0.35 1.00
```

chd is not correlated with any of the predictors. *adiposity*, *obesity*, *age* have higher correlations as compared to others which sort of makes sense considering the context.

```r
smoothScatter(saheart$adiposity, saheart$obesity)
```



```r
smoothScatter(saheart$adiposity, saheart$age)
```



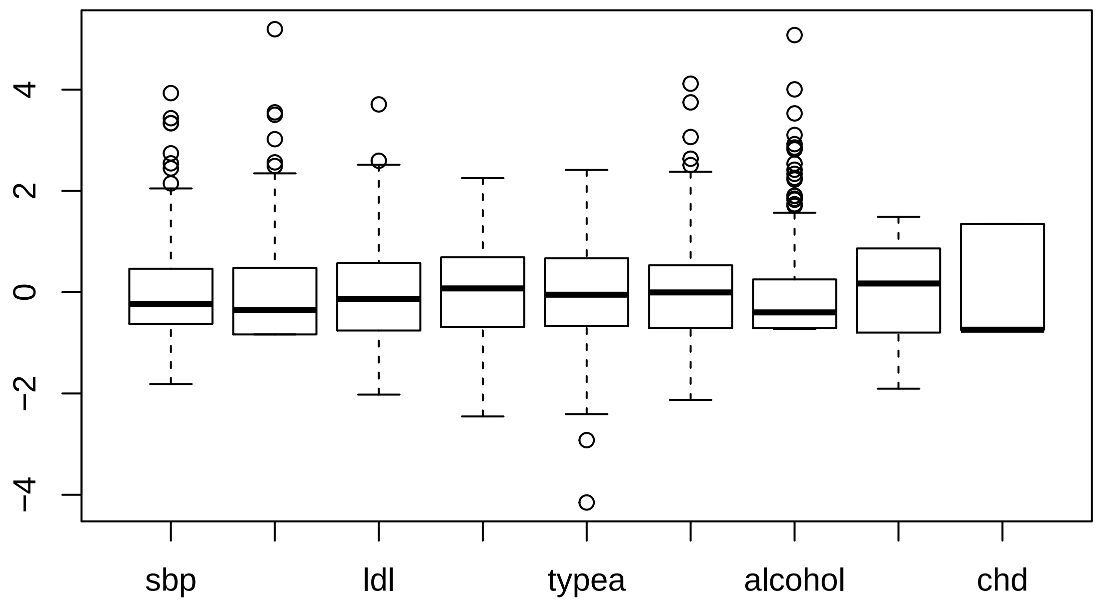
There is no missing data in the training set.

```r
any(is.na(saheart_training))
```

```
> [1] FALSE
```

Let's look for the presence of any outliers.

```r
boxplot(scale(saheart_training %>% dplyr::select(-famhist)))
```



There seem to be a few outlier points. But, let's leave them as is for now.

b. Logistic Regression and variable selection

Logistic Regression with all the predictors

```r
glm.fit <- glm(chd~., data=saheart_training, family="binomial")
summary(glm.fit)
```

```
>
> Call:
> glm(formula = chd ~ ., family = "binomial", data = saheart_training)
>
> Deviance Residuals:
>     Min       1Q   Median       3Q      Max
> -1.8634  -0.7938  -0.4391   0.9292   2.3888
>
> Coefficients:
>              Estimate Std. Error z value Pr(>|z|)
> (Intercept) -6.675624   1.982629  -3.367  0.00076 ***
> sbp          0.004035   0.008194   0.492  0.62243
> tobacco      0.068937   0.041765   1.651  0.09883 .
> ldl          0.175619   0.099796   1.760  0.07845 .
> adiposity    0.017191   0.047224   0.364  0.71583
> famhist      1.135890   0.323727   3.509  0.00045 ***
> typea        0.050532   0.018209   2.775  0.00552 **
> obesity     -0.046014   0.071775  -0.641  0.52147
> alcohol      0.004682   0.006296   0.744  0.45704
> age          0.040844   0.018309   2.231  0.02569 *
> ---
> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> (Dispersion parameter for binomial family taken to be 1)
>
>     Null deviance: 300.52  on 230  degrees of freedom
> Residual deviance: 238.60  on 221  degrees of freedom
> AIC: 258.6
>
> Number of Fisher Scoring iterations: 5
```

famhist, *typea* and *age* seem to be good predictors of *chd* according to their low p-values.

All subsets selection

```
library(bestglm)
```

```
> Loading required package: leaps
```

```
glm.fit.allsubsets <- bestglm(Xy = saheart_training, family = binomial, IC = "AIC", method = "exhaustiv
```

```
> Morgan-Tatar search since family is non-gaussian.
```

```
summary(glm.fit.allsubsets)
```

```
> Fitting algorithm:  AIC-glm
> Best Model:
>             df deviance
> Null Model 225 239.9422
> Full Model 230 300.5190
>
>   likelihood-ratio test - GLM
>
> data:  H0: Null Model vs. H1: Best Fit AIC-glm
> X = 60.577, df = 5, p-value = 9.237e-12
```

```
glm.fit.allsubsets$BestModel$coefficients
```

```
> (Intercept)     tobacco         ldl     famhist       typea         age
> -6.91292025  0.07652904  0.15606856  1.13809351  0.04984199  0.04614363
```

As seen above, from all subsets selection, the best model is the one with the predictors *tobacco*, *ldl*, *famhist*, *typea*, *obesity* and *age*.

c. Linear Discriminant Analysis

Since LDA assumes that the predictors are normally distributed, the categorical predictor *famhist* will need to be excluded.

```
library(MASS)
lda.fit <- lda(chd~sbp + tobacco + ldl + adiposity + typea + obesity + alcohol + age, data = saheart_tr
lda.fit
```

```
> Call:
> lda(chd ~ sbp + tobacco + ldl + adiposity + typea + obesity +
>     alcohol + age, data = saheart_training)
>
> Prior probabilities of groups:
>         0         1
> 0.6450216 0.3549784
>
> Group means:
>        sbp  tobacco      ldl adiposity    typea  obesity  alcohol      age
> 0 136.1208 2.577919 4.260403  23.93691 52.61745 25.64839 16.25732 38.72483
> 1 143.1585 5.065000 5.145732  27.52939 55.02439 26.42695 22.63037 49.37805
```

```
>
> Coefficients of linear discriminants:
>                    LD1
> sbp         0.0054003052
> tobacco     0.0797731919
> ldl         0.2297311261
> adiposity   0.0001864121
> typea       0.0413545289
> obesity    -0.0408712612
> alcohol     0.0051528907
> age         0.0444890599
```

From all subsets selection above, the selected predictors and their coefficients are $tobacco = 0.07652904$, $ldl = 0.15606856$, $typea = 0.04984199$ and $age = 0.04614363$. The respective coeffcients of LDA and best subsets are very close in value. $abp, adiposity$, and $alcohol$ were not selected by best subset selection and its coefficient in LDA is very close to 0.

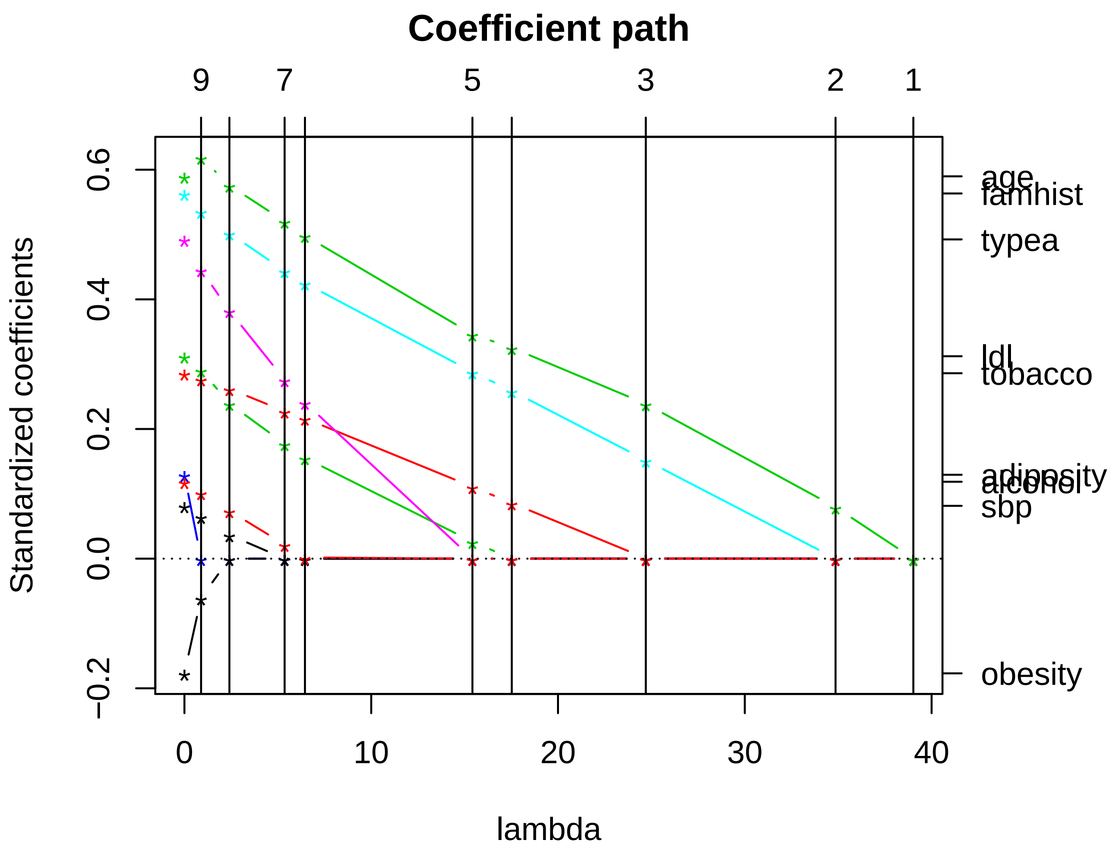
d. Logistic Regression with Lasso regularization

1. Produce and interpret plot of paths of the individual coefficients

```
library(glmpath)
```

```
> Loading required package: survival
```

```
glmpath.fit <- glmpath(y = saheart_training[, 10],
                       x = as.matrix(saheart_training[,-10]),
                       family = binomial)
par(mfrow=c(1,1), mar=c(4,4,4,8))
plot(glmpath.fit, xvar="lambda")
```

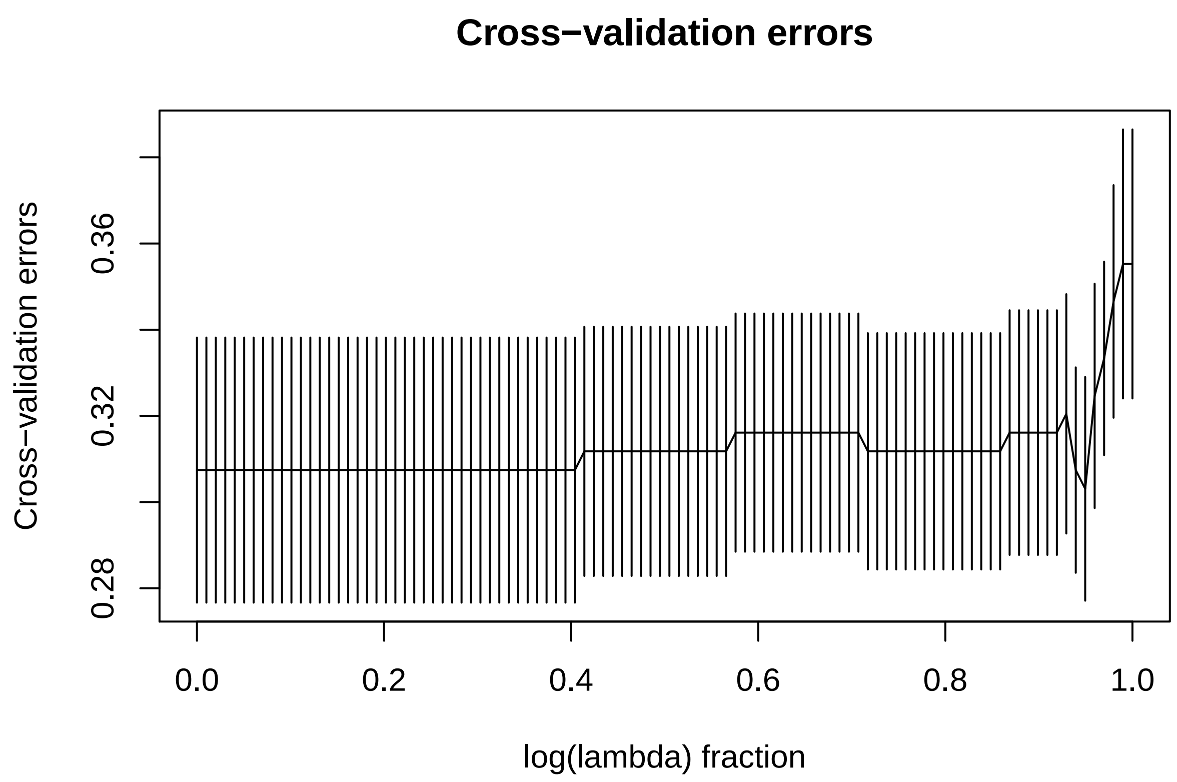


Different values of λ need to tried to find the one that minimizes error by cross-validation. Above, the estimated coefficients for different values of λ are plotted to see the predictors are being selected for each value, to see the effect of regularization. When λ is 0, there are more non-zero coefficients, but, as λ increases, the coefficients of many of the predictors become 0.

2. Produce the plot of regularization parameter vs cross-validated predicted error

```
cv.glmpath.fit <- cv.glmpath(x=(as.matrix(saheart_training[,-10])), y=saheart_training[,10], family=bin
```

```
> CV Fold 1
> CV Fold 2
> CV Fold 3
> CV Fold 4
> CV Fold 5
> CV Fold 6
> CV Fold 7
> CV Fold 8
> CV Fold 9
> CV Fold 10
```



The above function plots the cross validation errors for different values of λ.

3. Select regularization parameter, and the corresponding predictors

Let's look at the λ that has the minimum cross validation error

```
# Log lambda
cv.min_log_lambda <- cv.glmpath.fit$fraction[which.min(cv.glmpath.fit$cv.error)]
cv.min_log_lambda
```

```
> [1] 0.9494949
```

```
# lambda
exp(cv.min_log_lambda)
```

```
> [1] 2.584404
```

4. Fit the model with the selected predictors only on the full training set

```
pred.coef <- predict(glmpath.fit, s=cv.min_log_lambda, mode="norm.fraction", type="coefficients")
pred.coef
```

```
>                   Intercept         sbp    tobacco       ldl  adiposity
> 0.949494949495 -6.654188 0.003713788 0.06809507 0.1709175 0.01046364
>                    famhist      typea     obesity     alcohol        age
> 0.949494949495 1.114072 0.04864578 -0.03419457 0.004419863 0.04163242
> attr(,"s")
> [1] 0.9494949
> attr(,"fraction")
>         1
> 0.9494949
> attr(,"mode")
> [1] "norm.fraction"
```

e. Shrunken centroid model

1. Use cross-validation to select the best regularization parameter

```
library(pamr)
```

```
> Loading required package: cluster
```

```
pamr_training <- list(x=t(as.matrix(saheart_training[,-10])), y=saheart_training[,10])
pamr_validation <- list(x=t(as.matrix(saheart_validation[,-10])),y=saheart_validation[,10])


pamr.fit <- pamr.train(pamr_training)
```

```
> 123456789101112131415161718192021222324252627282930
```

```
print(pamr.fit$centroids)
```

```
>                     0           1
> sbp       136.1208054 143.1585366
> tobacco     2.5779195   5.0650000
> ldl         4.2604027   5.1457317
> adiposity  23.9369128  27.5293902
> famhist     0.3087248   0.6463415
> typea      52.6174497  55.0243902
> obesity    25.6483893  26.4269512
> alcohol    16.2573154  22.6303659
> age        38.7248322  49.3780488
> attr(,"scaled:scale")
> y
>   0   1
> 149  82
```

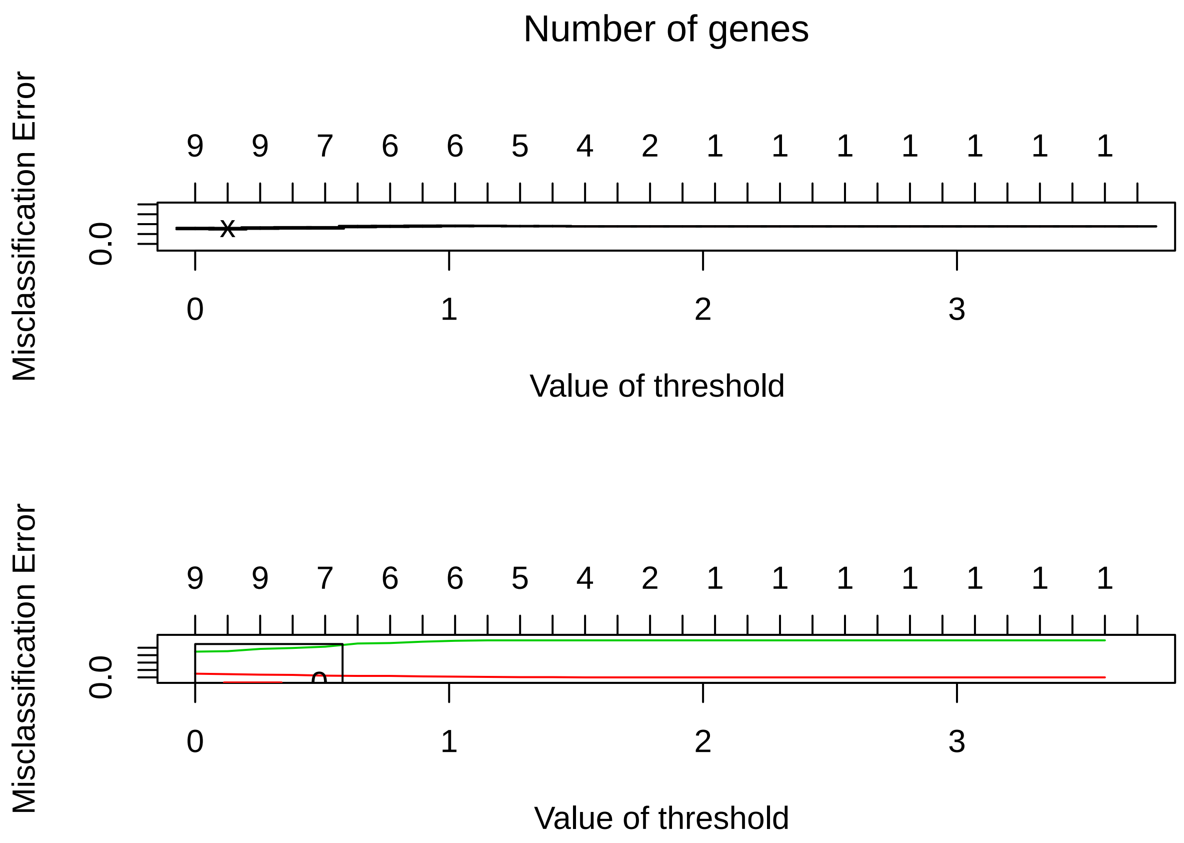
```
pamr.cv.fit <- pamr.cv(pamr.fit, pamr_training)
```

```
> 12Fold 1 :123456789101112131415161718192021222324252627282930
> Fold 2 :123456789101112131415161718192021222324252627282930
> Fold 3 :123456789101112131415161718192021222324252627282930
> Fold 4 :123456789101112131415161718192021222324252627282930
> Fold 5 :123456789101112131415161718192021222324252627282930
> Fold 6 :123456789101112131415161718192021222324252627282930
> Fold 7 :123456789101112131415161718192021222324252627282930
> Fold 8 :123456789101112131415161718192021222324252627282930
> Fold 9 :123456789101112131415161718192021222324252627282930
> Fold 10 :123456789101112131415161718192021222324252627282930
```

```
pamr.cv.fit
```

```
> Call:
> pamr.cv(fit = pamr.fit, data = pamr_training)
>    threshold nonzero errors
> 1  0.000     9       72
> 2  0.128     9       71
> 3  0.256     9       74
> 4  0.384     8       75
> 5  0.512     7       75
> 6  0.640     7       81
> 7  0.768     6       82
> 8  0.896     6        83
> 9  1.024     6        84
> 10 1.152     5        84
> 11 1.279     5        83
> 12 1.407     5        83
> 13 1.535     4        82
> 14 1.663     3        82
> 15 1.791     2        82
> 16 1.919     1        82
> 17 2.047     1        82
> 18 2.175     1        82
> 19 2.303     1        82
> 20 2.431     1        82
> 21 2.559     1        82
> 22 2.687     1        82
> 23 2.815     1        82
> 24 2.943     1        82
> 25 3.071     1        82
> 26 3.199     1        82
> 27 3.327     1        82
> 28 3.455     1        82
> 29 3.583     1        82
> 30 3.711     0        82
```

```r
par(mar=c(1,1,1,1))
pamr.plotcv(pamr.cv.fit)
```



For some reason, I'm unable to get a good represenation of the graphs in this document. However, upon zooming into it, I observed that the misclassification error is small for small values of threshold. Unlike for *chd* = 0, as the threshold increases, the misclassification of *chd* = 1 samples increases. I'll choose a threshold of 0.6 as it looks like it has the largest shrinkage without a resulting increase in error.

Let's look the confusion matrix:

```r
pamr.confusion(pamr.cv.fit, threshold=.6)
```

```
>     0 1 Class Error rate
> 0 143 6       0.04026846
> 1  75 7       0.91463415
> Overall error rate= 0.349
```

2. Refit the model with the selected regularization parameter

```r
# Refit the classifier on the full dataset, but using the threshold
pamr.fit <- pamr.train(pamr_training, threshold=0.6)
```

```
> 1
```

```r
pamr.fit
```

```
> Call:
> pamr.train(data = pamr_training, threshold = 0.6)
>   threshold nonzero errors
> 1 0.6       7       74
```

3. Visualize the centroids of the selected model

```r
# pamr.plotcen(pamr.fit, pamr_training, threshold=0.6)

# Could not stitch the plot produced by the above command.
# I've attached it in the folder instead with the name "p_4_e_3.jpg"
```

f. Classifier performance

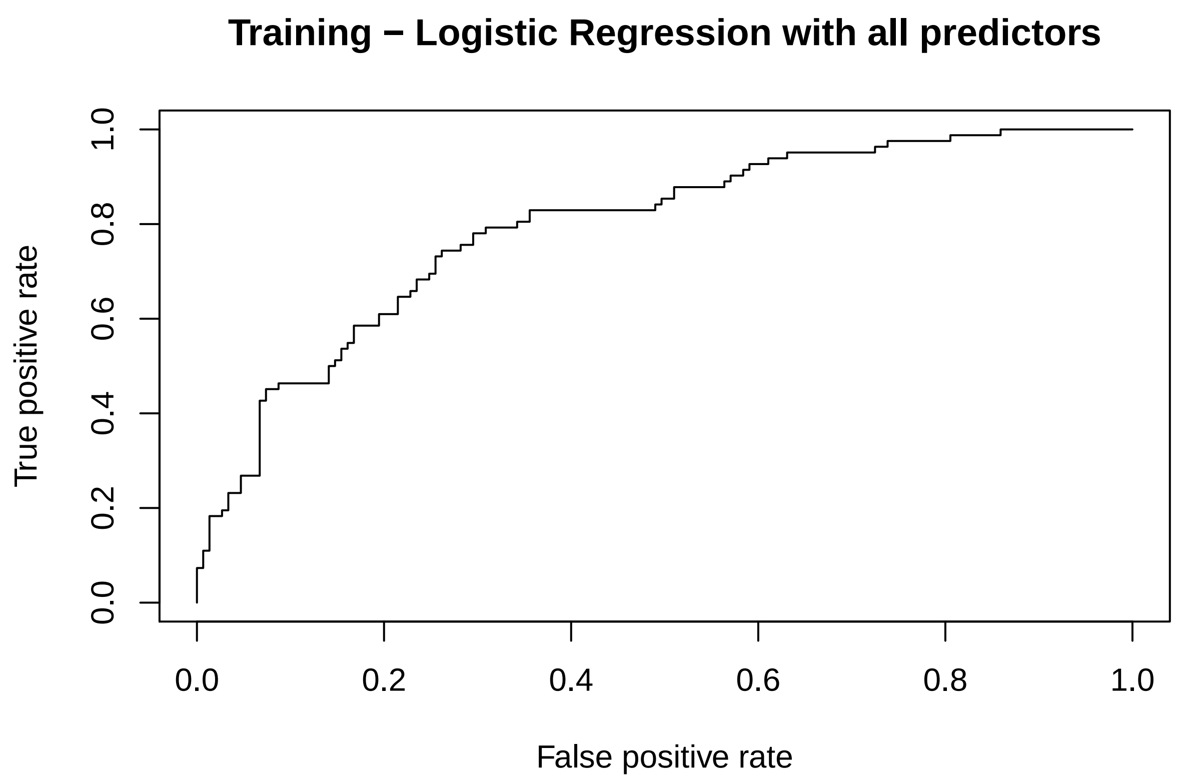
1. Evaluation on the training and validation sets To check the performance of the classifiers, we plot ROC curves to compare their areas. Higher the area, better the classifier.

For logistic regression with all predictors:

```r
library(ROCR)
```

```
> Loading required package: gplots

>
> Attaching package: 'gplots'

> The following object is masked from 'package:stats':
>
>     lowess
```

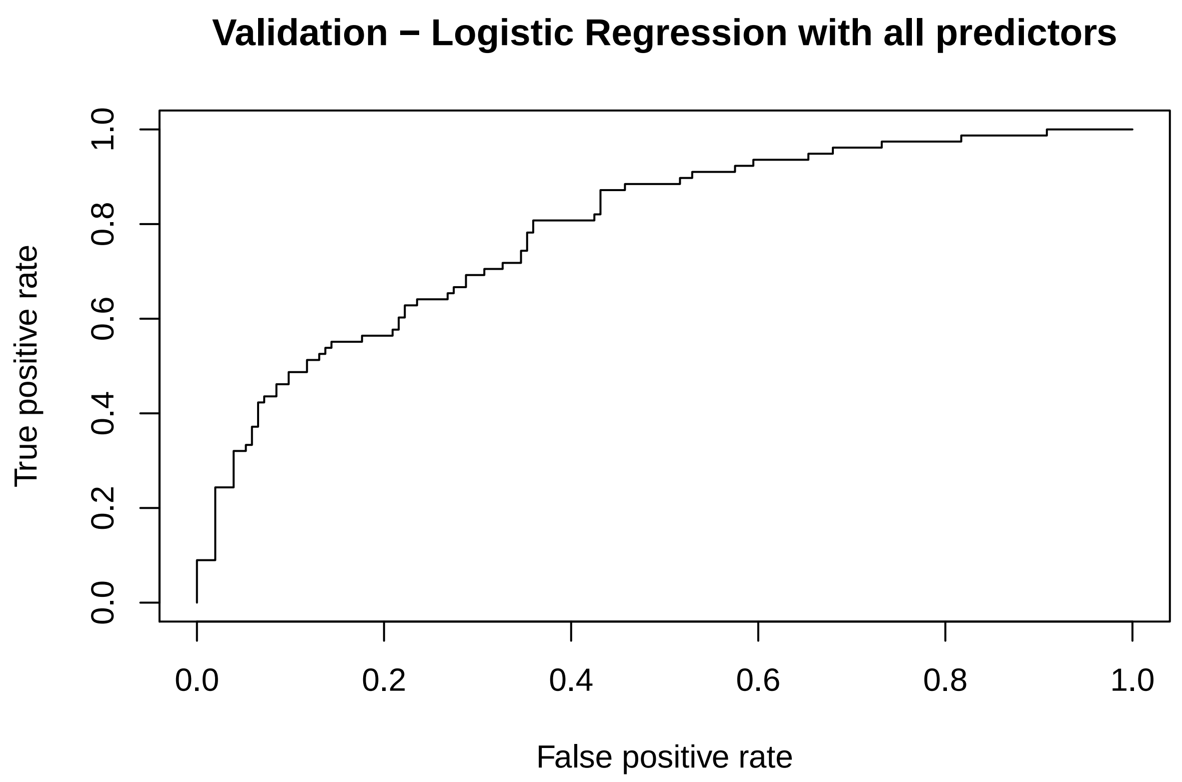
```r
# Training
glm.pred <- predict(glm.fit, saheart_training %>% dplyr::select(-chd), type="response")
glm.pred <- prediction(glm.pred, saheart_training %>% pull(chd))
glm.perf <- performance(glm.pred, 'tpr', 'fpr')
plot(glm.perf, colorize=F, main="Training - Logistic Regression with all predictors")
```



```
# Area
cat("Area ", unlist(attributes(performance(glm.pred, "auc"))$y.values))
```

```
> Area  0.7933377
```

```
# Validation
glm.pred.validation <- predict(glm.fit, saheart_validation %>% dplyr::select(-chd), type="response")
glm.pred.validation <- prediction(glm.pred.validation, saheart_validation %>% pull(chd))
glm.perf.validation <- performance(glm.pred.validation, 'tpr', 'fpr')
plot(glm.perf.validation, colorize=F, main="Validation - Logistic Regression with all predictors")
```

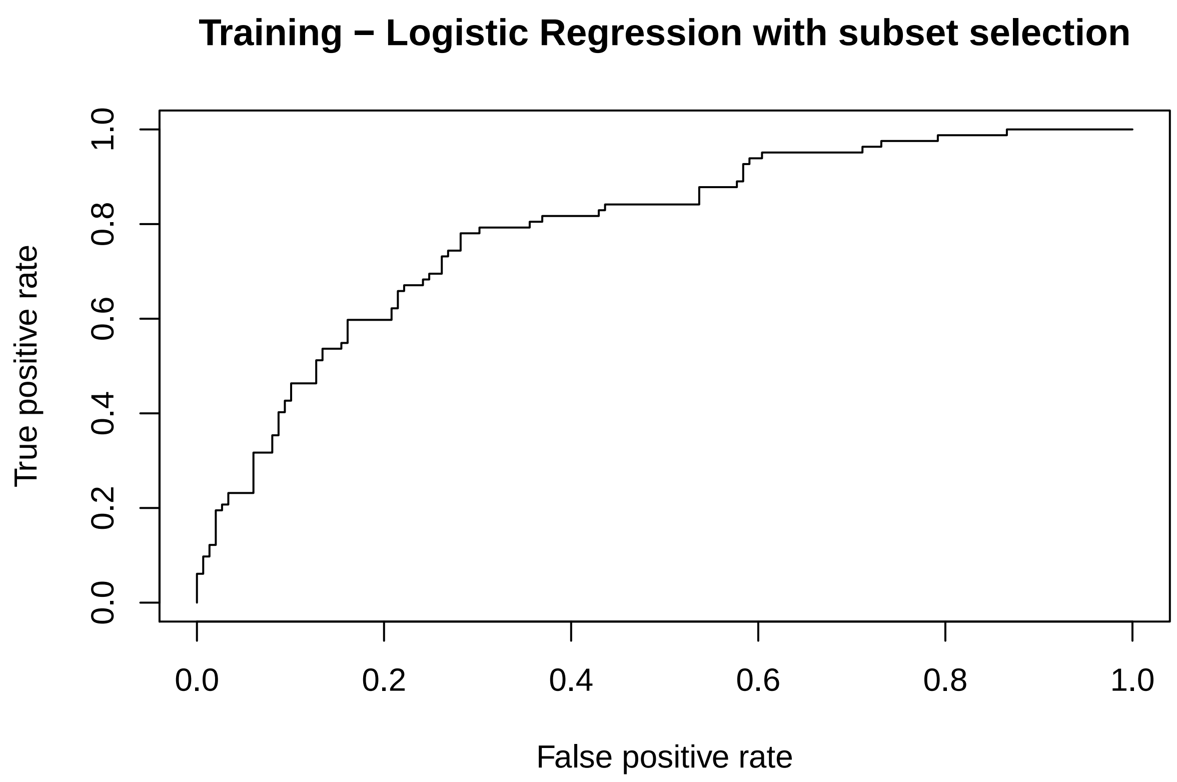


```r
# Area
cat("Area ", unlist(attributes(performance(glm.pred.validation, "auc"))$y.values))
```

```
> Area  0.7900117
```

For logistic regression with subset selection:

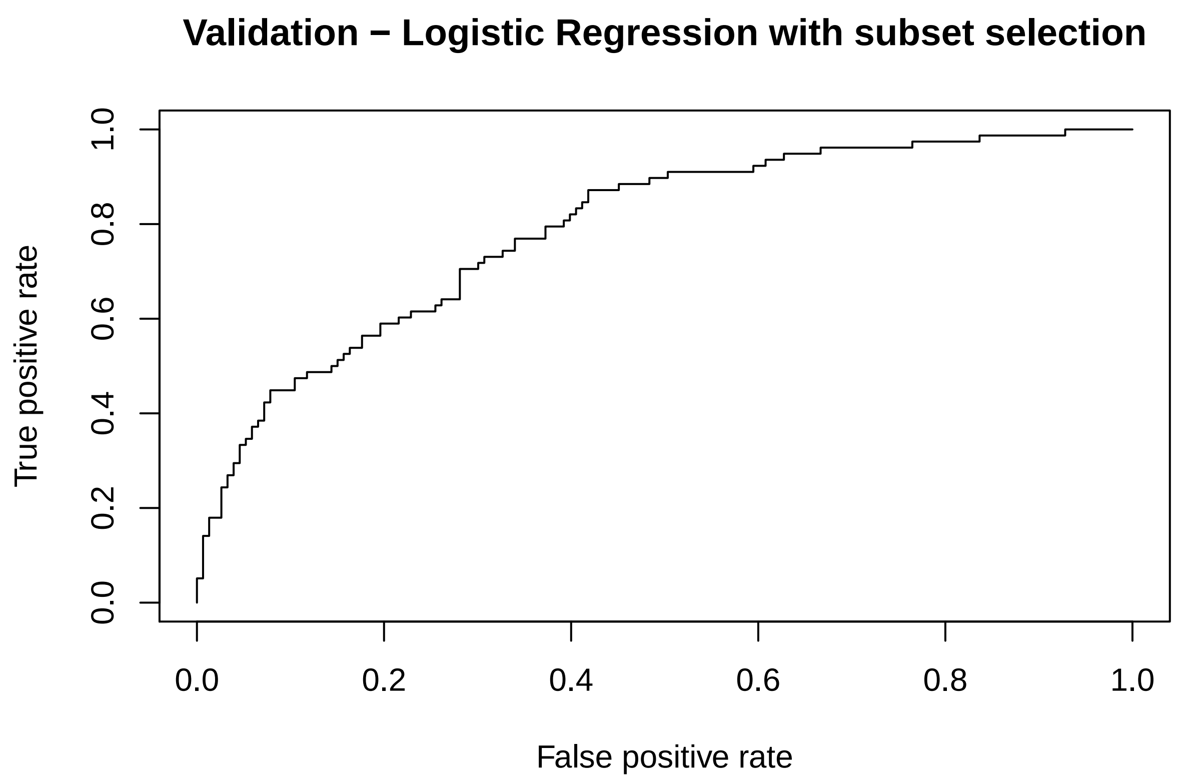
```r
# Training
glm.pred.allsubsets <- predict(glm.fit.allsubsets$BestModel, saheart_training %>% dplyr::select(-chd),
type="response")
glm.pred.allsubsets <- prediction(glm.pred.allsubsets,saheart_training %>% pull(chd))
glm.perf.allsubsets <- performance(glm.pred.allsubsets, 'tpr', 'fpr')
plot(glm.perf.allsubsets, colorize=F, main="Training - Logistic Regression with subset selection")
```



```r
# Area
cat("Area ", unlist(attributes(performance(glm.pred.allsubsets, "auc"))$y.values))
```

```
> Area  0.7909642
```

```r
# Validation
glm.pred.allsubsets.validation <- predict(glm.fit.allsubsets$BestModel, saheart_validation %>% dplyr::s
glm.pred.allsubsets.validation <- prediction(glm.pred.allsubsets.validation, saheart_validation %>% pul
glm.perf.allsubsets.validation <- performance(glm.pred.allsubsets.validation, 'tpr', 'fpr')
plot(glm.perf.allsubsets.validation, colorize=F, main="Validation - Logistic Regression with subset sel
```

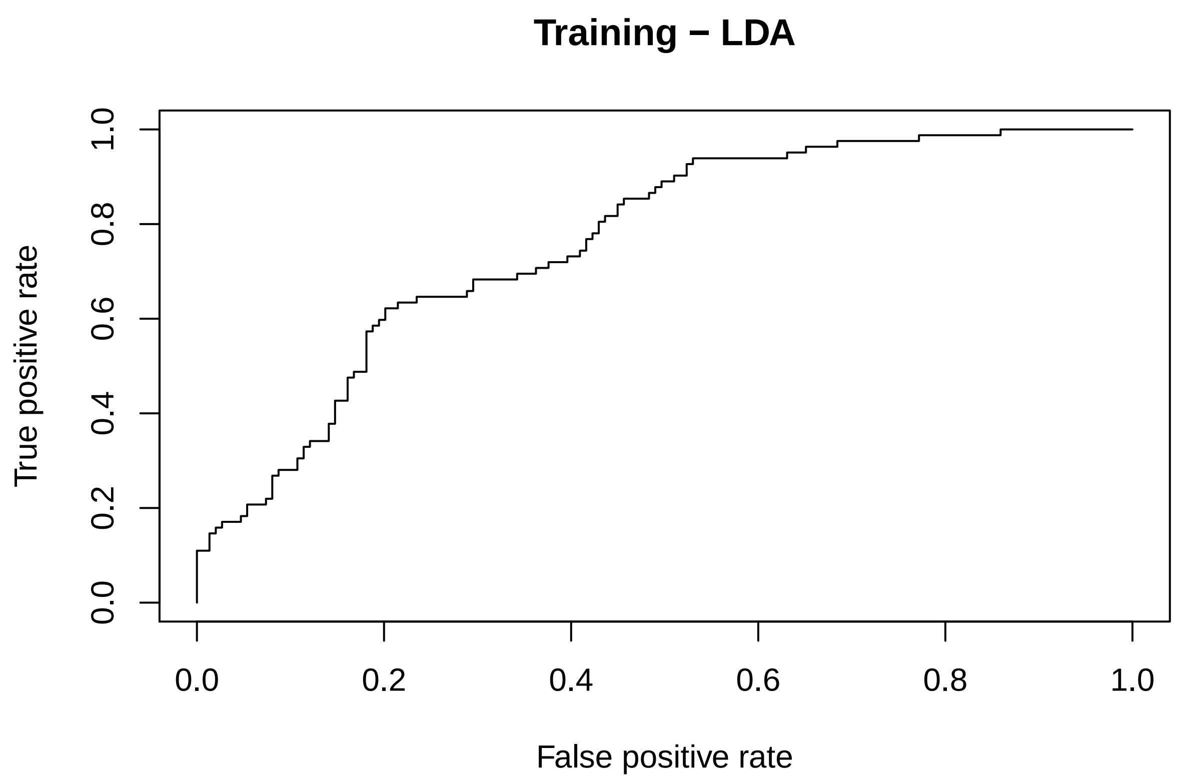


```
# Area
cat("Area ", unlist(attributes(performance(glm.pred.allsubsets.validation, "auc"))$y.values))
```

```
> Area  0.7894252
```

For LDA classifier:

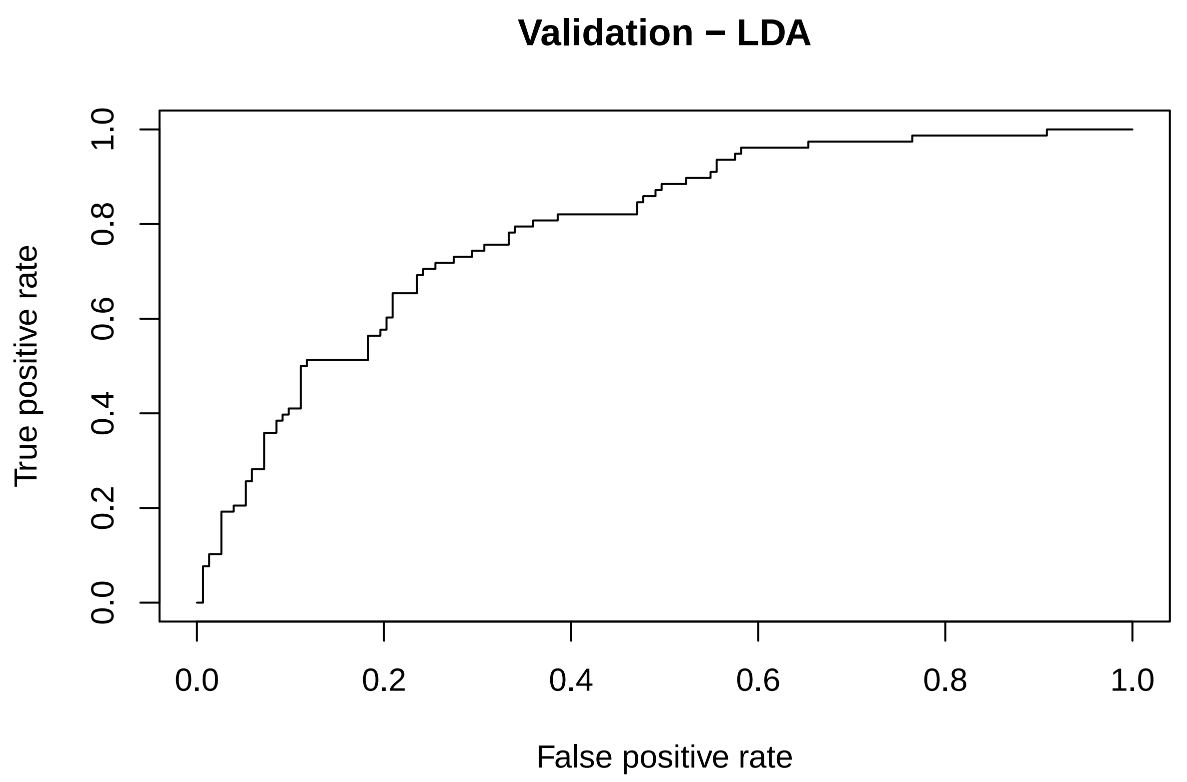
```
# Training
lda.pred <- predict(lda.fit, saheart_training %>% dplyr::select(-chd))$posterior[,2]
lda.pred <- prediction(lda.pred, saheart_training %>% pull(chd))
lda.perf <- performance(lda.pred, "tpr", "fpr")
plot(lda.perf, colorize=F, main="Training - LDA")
```



```
# Area
cat("Area ", unlist(attributes(performance(lda.pred, "auc"))$y.values))
```

```
> Area  0.7628908
```

```
# Validation
lda.pred.valid <- predict(lda.fit, saheart_validation %>% dplyr::select(-chd))$posterior[,2]
lda.pred.valid <- prediction(lda.pred.valid, saheart_validation %>% pull(chd))
lda.perf.valid <- performance(lda.pred.valid, "tpr", "fpr")
plot(lda.perf.valid, colorize=F, main="Validation - LDA")
```

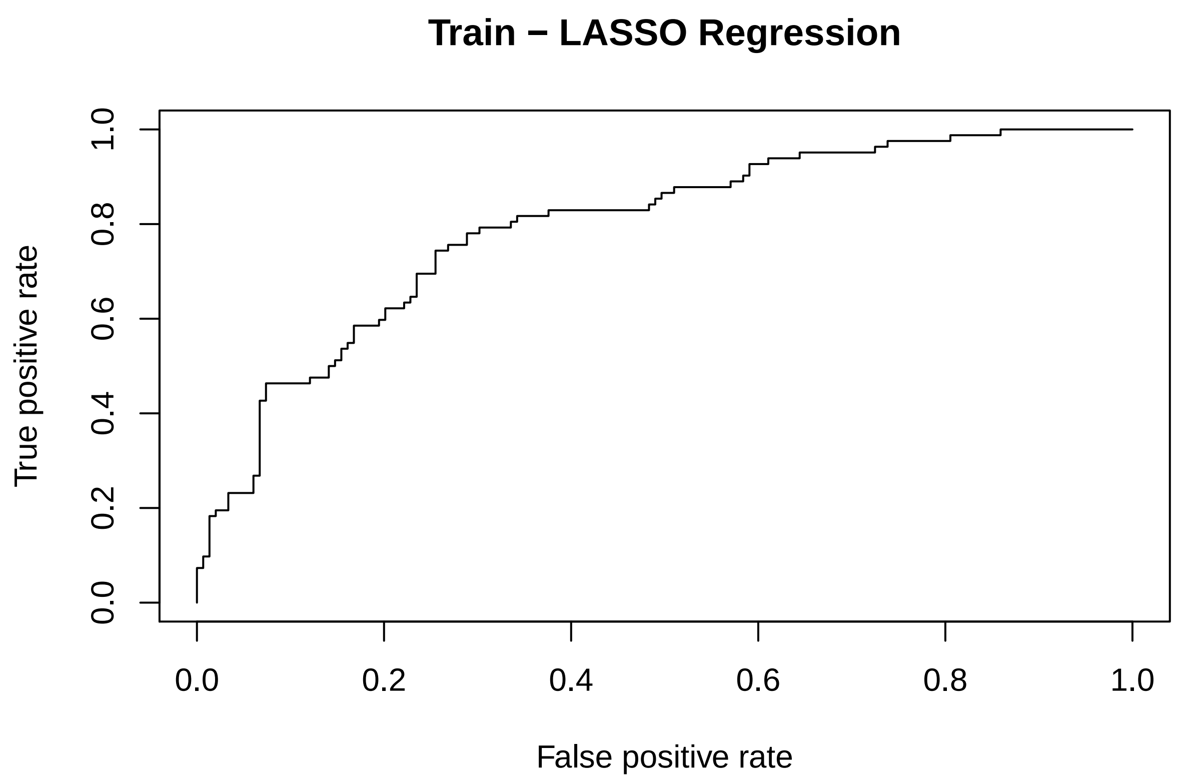


```
# Area
cat("Area ", unlist(attributes(performance(lda.pred.valid, "auc"))$y.values))
```

```
> Area  0.7900955
```

For Lasso regression:

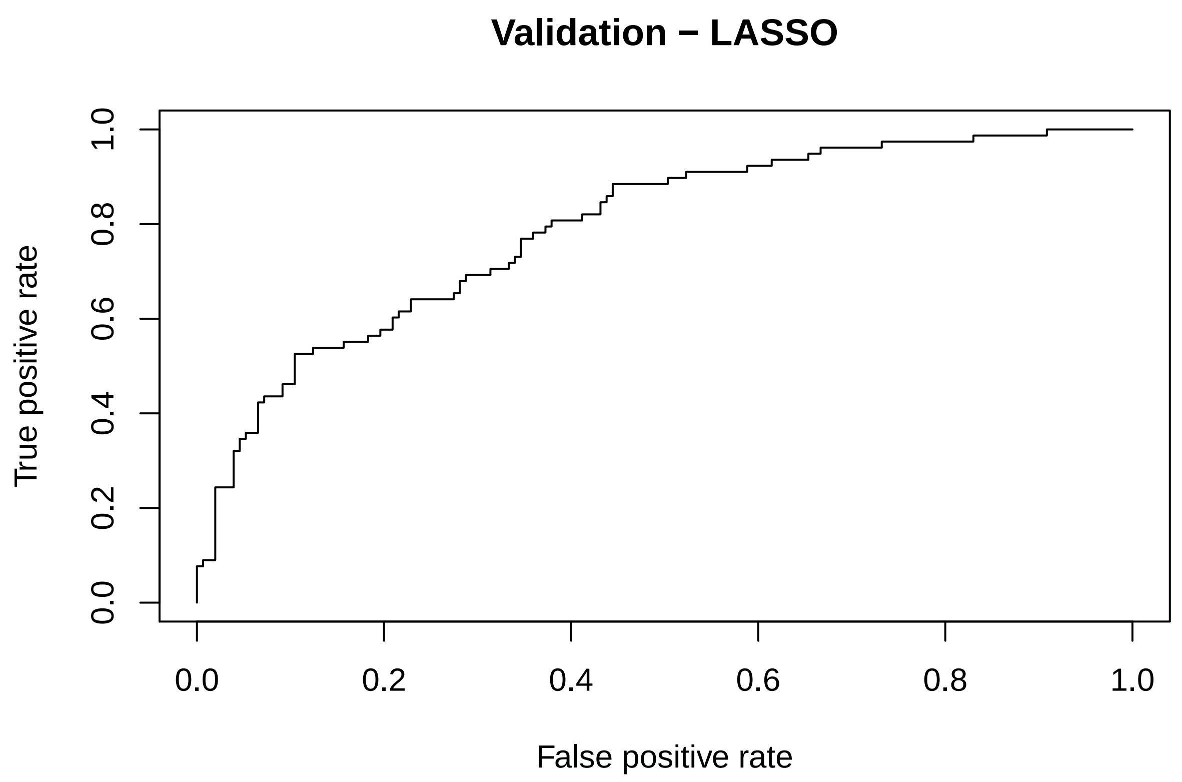
```
# Training
glmpath.pred <- predict(glmpath.fit,newx=as.matrix(saheart_training %>% dplyr::select(-chd)),s=cv.min_l
glmpath.pred <- prediction(predictions=glmpath.pred, labels=saheart_training %>% pull(chd))
glmpath.perf <- performance(glmpath.pred, "tpr", "fpr")
plot(glmpath.perf, colorize=F, main="Train - LASSO Regression")
```



```
# Area
cat("Area ", unlist(attributes(performance(glmpath.pred, "auc"))$y.values))
```

```
> Area  0.7935014
```

```
# Validation
glmpath.pred.validation <- predict(glmpath.fit,newx=as.matrix(saheart_validation %>% dplyr::select(-chd
s=cv.min_log_lambda,mode="norm.fraction",type="response")
glmpath.pred.validation <- prediction(predictions=glmpath.pred.validation,
labels=saheart_validation %>% pull(chd))
glmpath.perf.valid <- performance(glmpath.pred.validation, "tpr", "fpr")
plot(glmpath.perf.valid, colorize=F, main="Validation - LASSO")
```

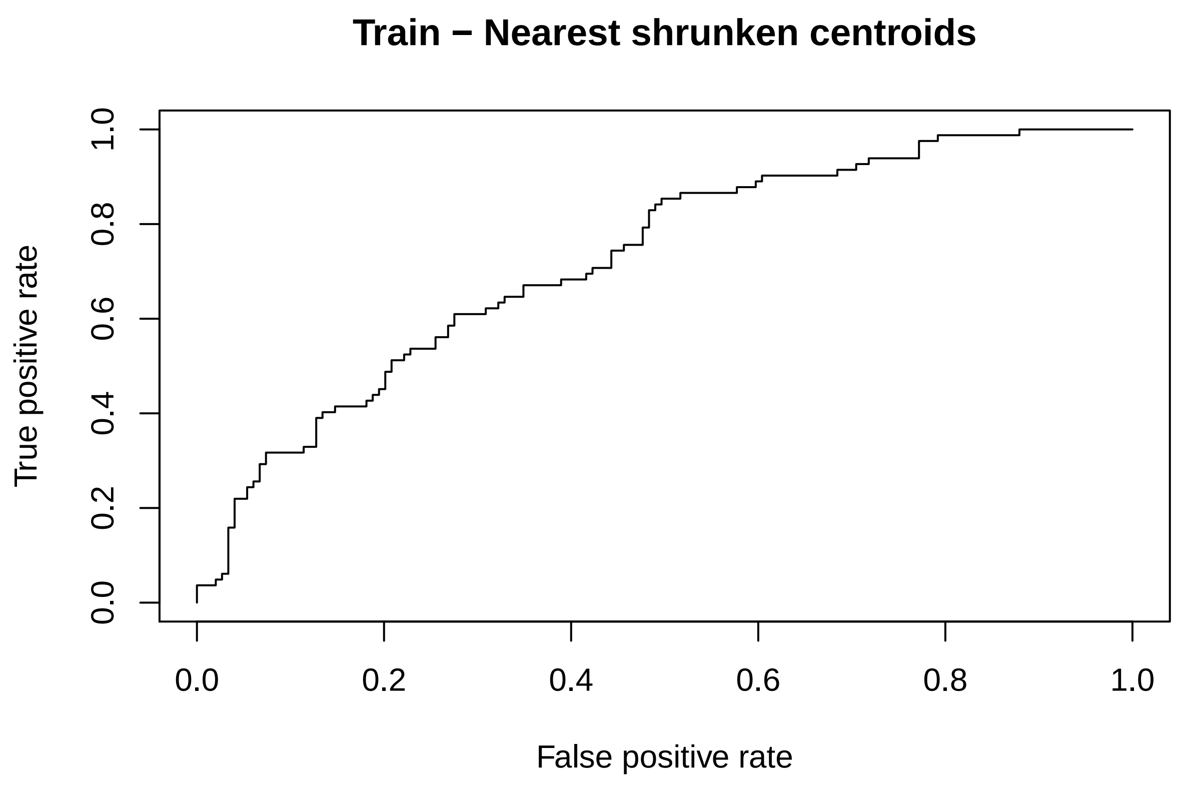


```
# Area
cat("Area ", unlist(attributes(performance(glmpath.pred.validation, "auc"))$y.values))
```

```
> Area  0.7902631
```

For nearest shrunked centroids:

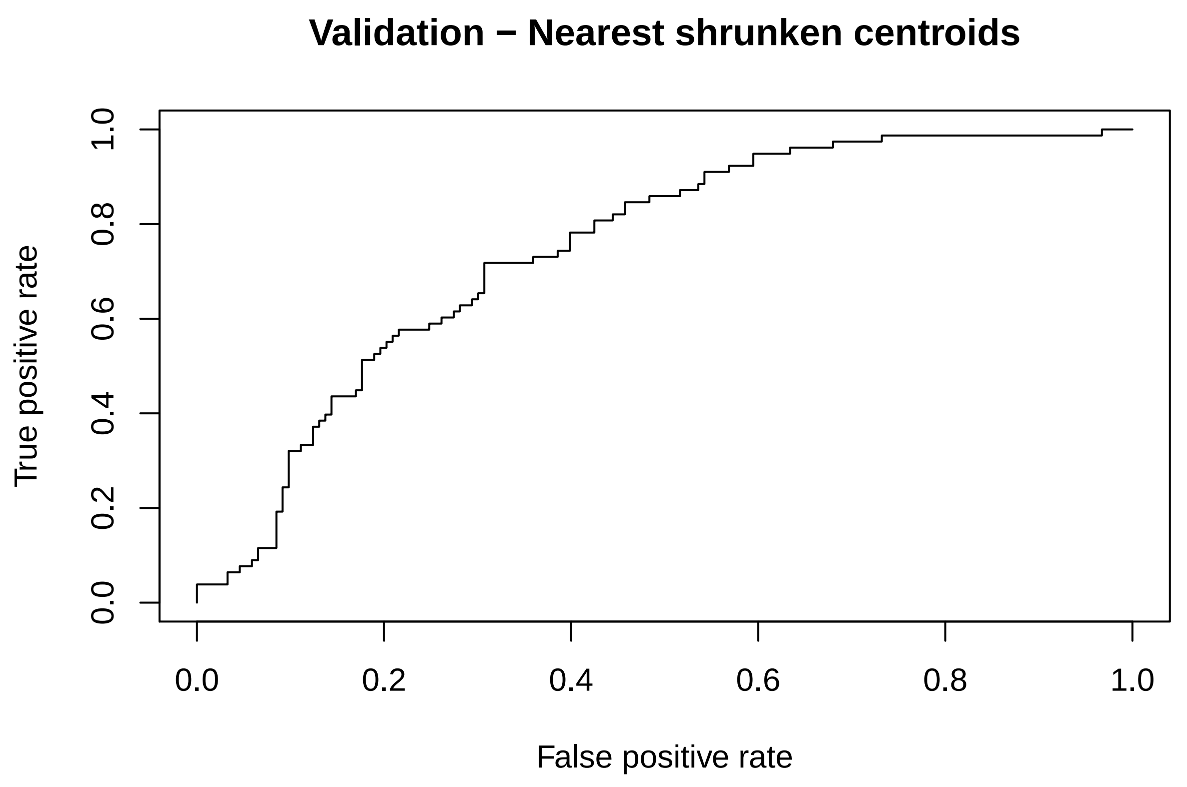
```
# Train
pamr.pred <- pamr.predict(pamr.fit, newx=pamr_training$x, threshold=0.6, type="posterior")[,2]
pamr.pred <- prediction(predictions=pamr.pred, labels=saheart_training %>% pull(chd))
pamr.perf <- performance(pamr.pred, "tpr", "fpr")
plot(pamr.perf, colorize=F, main="Train - Nearest shrunken centroids")
```



```
# Area
cat("Area ", unlist(attributes(performance(pamr.pred, "auc"))$y.values))
```

```
> Area  0.7240956
```

```
# Validation
pamr.pred.validation <- pamr.predict(pamr.fit, newx=pamr_validation$x, threshold=0.6, type="posterior")
pamr.pred.validation <- prediction(predictions=pamr.pred.validation, labels=saheart_validation %>% pull
pamr.perf.validation <- performance(pamr.pred.validation, "tpr", "fpr")
plot(pamr.perf.validation, colorize=F, main="Validation - Nearest shrunken centroids")
```



```r
# Area
cat("Area ", unlist(attributes(performance(pamr.pred.validation, "auc"))$y.values))
```

```
> Area  0.7467739
```

g. Summary

From the ROC curves, it can be seen that the area under the curves for training sets is greater than the ones for validation sets, as it should be.

On both the training and validation sets, logistic regression with lasso regularization has the best performance. It has fewer predictors and can take into account categorical predictors as well unlike LDA.