Assignment 4: The Model to Beat

4/19/24

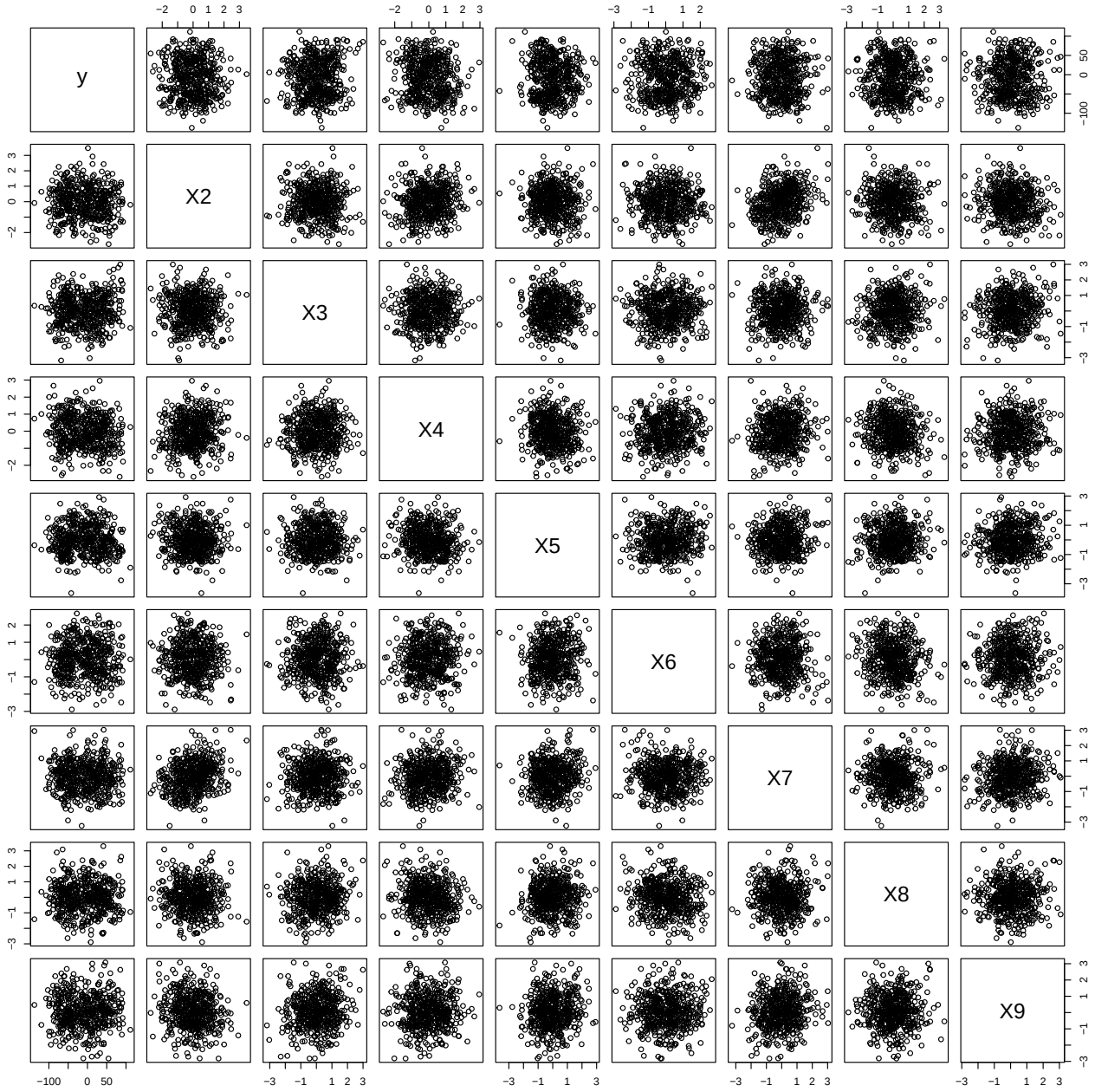
Why I am the best

I don't think I need to explain this. Maybe I will explain it more once I get back from my ego trip.

Exploratory Data Analysis

First, I plot the data a little to understand characteristics about it. Plotting a scatterplot of some of the columns

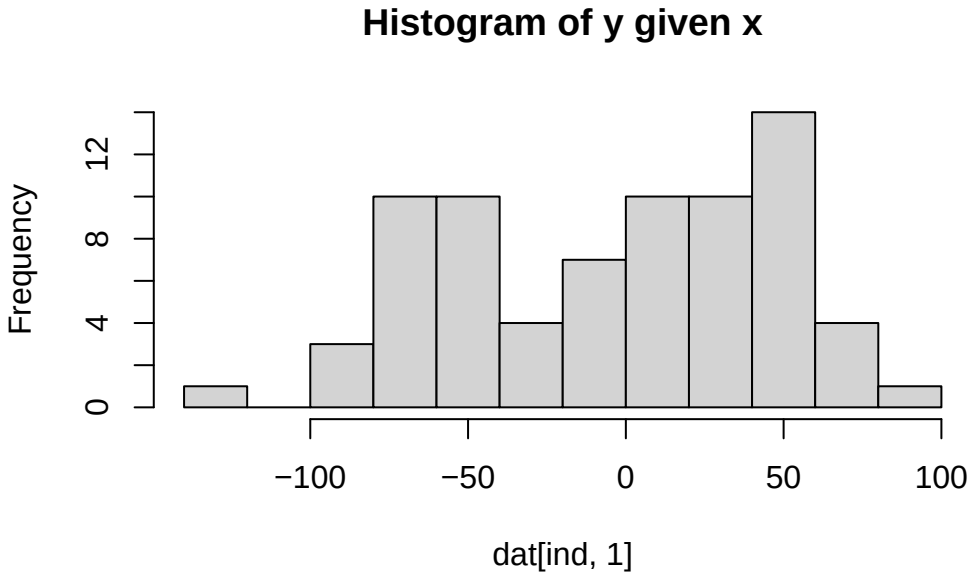
```
dat <- read.csv("assign4_train.csv")
plot(dat[ ,c(1,3:10)])
```



Only the top row is important really. Seeing the graphs zoomed out can actually be useful since it allows us to focus on patterns, rather than individual data points. The scatterplot with `y` and `X6` is particularly interesting since it seems to suggest two different types of cloud of data. Such a pattern seems like it might be present in the full top row.

Let's do something more specific now. Let's fix a small window of values of `X6`, let's say around -1, and look at the corresponding values of y:

```
ind <- (dat$X6 > -1.3 & dat$X6 < -.7)
hist(dat[ind, 1], main = "Histogram of y given x")
```



The above plot then suggests that the distribution of $y|x$ is possibly a mixture distribution with two mixture components. This exploration then leads us to the following model.

Mixture of Gaussian Response Regression

I am now motivated to build the following model from the previous histogram. Given $x_i \in \mathbb{R}^p$, it seems that the response are bimodal. with possibly its own regression coefficient. So:

$$y_i|x_i \sim pN(x_i^T\beta_1, \sigma_1^2) + (1-p)N(x_i^T\beta_2, \sigma_2^2)$$

That is, each response is coming from a two-component mixture where the means of the mixture components are different. Notice that if I now assume the existence of latent variables, Z_i, then I have the following distributions:

$$\Pr(Z_i = 1) = p \quad \text{and } \Pr(Z_i = 2) = 1 - p\,,$$

and

$$Y_i|x_i, Z_i = c \sim N(x_i^T\beta_c, \sigma_c^2)\,.$$

Using the above, we have the conditional distribution of Z_i given x_i:

$$\Pr(Z_i = c|x_i, \theta_{(k)}) = \frac{p\, f_1(y_i|x_i, \beta_{c,(k)}, \sigma^2_{c,(k)})}{p\, f_1(y_i|x_i, \beta_{1,(k)}, \sigma^2_{1,(k)}) + (1-p)\, f_2(y_i|x_i, \beta_{2,(k)}, \sigma^2_{2,(k)})} =: \gamma_{i,c,(k)}\,.$$

Thus, every observation is in a class, and each class has its own regression line. Our goal is to estimate the $\theta = (p, \beta_1, \beta_2, \sigma_1^2, \sigma_2^2)$ where the log-likelihood is $\log f(\mathbf{y}|X, \theta)$. We can use EM algorithm to estimate θ using the missing data approach. Recall that, in the EM algorithm then finds a minorizing function $q(\theta|\theta_{(k)})$ such that

$$\log f(\mathbf{y}|X,\theta) \geq q(\theta|\theta_{(k)}) + \text{constant}\,,$$

where we know that

$$\begin{aligned} q(\theta|\theta_{(k)}) &= \sum_{i=1}^{n} \mathrm{E}_{Z_i}\left(\log f(y_i|x_i,z_i,\theta) \mid x_i,\theta_{(k)}\right) \\ &= \sum_{i=1}^{n}\left[\log(pf_1(y_i|x_i))\gamma_{i,1,(k)} + \log((1-p)f_2(y_i|x_i))\gamma_{i,2,(k)}\right]\,. \end{aligned}$$

Using the above minorization function, we can then use an iterated algorithm to obtain the MLE of θ. However, I will not use MLE, but use a penalty.

Adding lasso penalty

Instead of obtaining the MLE of θ, I was to obtain a penalized MLE using a lasso penalty (I've used lasso only because I didn't have time to test all values of α in the Bridge penalty). The final objective function I want to solve is:

$$\arg\max_{\theta} \underbrace{\left\{\log f(\mathbf{y}|X,\theta) - \frac{\lambda_1}{\sigma_1^2}\sum_{i=1}^{p}|\beta_{1,i}| - \frac{\lambda_2}{\sigma_2^2}\sum_{i=1}^{p}|\beta_{2,i}|\right\}}_{Q(\theta)}$$

Luckily, I note that I already have a minorization function the first part, and also from our MM algorithm from Bridge regression notes. That is

$$Q(\theta) \geq \text{const} + q(\theta|\theta_{(k)}) - \frac{\lambda_1}{\sigma_1^2}\sum_{j=1}^{p}\frac{\beta_{1,j}^2}{|\beta_{1,j,(k)}|} - \frac{\lambda_2}{\sigma_2^2}\sum_{j=1}^{p}\frac{\beta_{2,j}^2}{|\beta_{2,j,(k)}|} =: t(\theta|\theta_{(k)})\,.$$

Then we see that $t(\theta|\theta_{(k)})$ is a minorizing function and we can use the MM algorithm to solve optimization method. This we can do by taking derivatives and setting to zero. I won't show all the work, but below are the final iterates (this is homework for you!).

Let $\Gamma_{c,(k)}$ denote the $n\times n$ matrix of $\gamma_{i,c,(k)}$, $i = 1,2,\ldots,n$, and $M_{(k)}$ is the $p\times p$ diagonal matrix of $|\beta_{2,j,(k)}|^{-1}$ Then

$$\begin{aligned} \beta_{c,(k+1)} &= (X^T\Gamma_{c,(k)}X + \lambda_c M_{(k)})^{-1}X^T\Gamma_{c,(k)}y \\ \sigma^2_{c,(k+1)} &= \frac{(y - X\beta_{(k+1)})^T\Gamma_{c,(k)}(y - X\beta_{(k+1)}) + 2\lambda_1\sum_{j=1}^{p}|\beta_{1,j}| + 2\lambda_2\sum_{j=1}^{p}|\beta_{2,j}|}{\sum_{i=1}^{n}\gamma_{i,c,(k)}} \\ p_{(k+1)} &= \sum_{i=1}^{n}\frac{\gamma_{i,c,(k)}}{n}\,. \end{aligned}$$

The above then sets it up for us. One thing to decide are the starting values. I visually note from the boxplot that one mode in y is below 0 and another that is above 0. So I will use this to determine starting values. Starting values will be critical since the objective function is highly non-concave.

Implementation on the data

Below is the main function

```
######################################################################
## Conditional mixture of Gaussians fix
######################################################################

# Y | X ~ p N(X beta_1, sigma1^2) + (1-p) N(X beta_2, sigma2^2)
# Will use penalty for this beta1 and beta2, separately

log.like <- function(y, X, beta1, beta2, sig1, sig2, pi1, lam1, lam2)
{

  track <- sum(log(pi1*dnorm(y, X%*%beta1, sd = sqrt(sig1)) +
                     (1-pi1)*dnorm(y, X%*%beta2, sd = sqrt(sig2))))
  track <- track - lam1*sum(abs(beta1))/(sig1) - lam2*sum(abs(beta2))/(sig2)

  if(track == -Inf) track = -1e+30
  return(track)
}

fitTrain <- function(y, X, lam1, lam2)
{
  n <- length(y)
  p <- dim(X)[2]

  coefs <- lm(y ~ X - 1)$coef
  index <- y < 0

  fit1 <- lm(y[index] ~ X[index, ] - 1)
  fit2 <- lm(y[!index] ~ X[!index, ] - 1)
  beta1 <- fit1$coefficients
  beta2 <- fit2$coefficients

  sig1 <- var(fit1$residuals)
  sig2 <- var(fit2$residuals)
  pi1 <- sum(index)/n
```

```r
diff <- 100
tol <- 1e-3
iter <- 0
while(diff > tol)
{
  iter <- iter + 1
  current <- log.like(y, X, beta1, beta2, sig1, sig2, pi1, lam1, lam2)
  # E step
  temp1 <- log(pi1) + dnorm(y, X%*%beta1, sd = sqrt(sig1), log = TRUE)
  temp2 <- log(1-pi1) + dnorm(y, X%*%beta2, sd = sqrt(sig2), log = TRUE)
  M <- pmin(abs(temp1), abs(temp2))

  temp1 <- exp(temp1 + M)
  temp2 <- exp(temp2 + M)
  gamma_1 <- as.numeric(temp1/(temp1 + temp2))
  Gamma1 <- diag(gamma_1, n)

  gamma_2 <- 1 - gamma_1
  Gamma2 <- diag(gamma_2, n)

  # I will only do lasso alpha = 1
  M1 <- diag(abs(as.numeric(beta1))^(-1), p)
  M2 <- diag(abs(as.numeric(beta2))^(-1), p)
  #M step
  beta1 <- qr.solve( t(X) %*% Gamma1 %*% X + lam1*M1, tol = 1e-20)%*% t(X) %*% Gamma1 %*% y
  beta2 <- qr.solve( t(X) %*% Gamma2 %*% X + lam2*M2, tol = 1e-20)%*% t(X) %*% Gamma2 %*% y

  sig1 <- (sum((y - X %*%beta1)^2*gamma_1)  + 2*lam1* sum(abs(as.numeric(beta1)) ) )/sum(gamma_1)
  sig2 <- (sum((y - X %*%beta2)^2*gamma_2)  + 2*lam2* sum(abs(as.numeric(beta2)) ) )/sum(gamma_2)

  pi1 <- mean(gamma_1)

  update <- log.like(y, X, beta1, beta2, sig1, sig2, pi1, lam1, lam2)
  diff <- norm(current - update, "2")
  #print(update)
}

temp1 <- pi1* dnorm(y, X%*%beta1, sd = sqrt(sig1))
temp2 <- (1-pi1)* dnorm(y, X%*%beta2, sd = sqrt(sig2))

gamma_1 <- as.numeric(temp1/(temp1 + temp2))
Gamma1 <- diag(gamma_1, n)
```

```
  gamma_2 <- 1 - gamma_1
  Gamma2 <- diag(gamma_2, n)
  gamma_ick <- cbind(gamma_1, gamma_2)

  rtn <- list("beta1" = beta1, "beta2" = beta2, "sig1" = sig2,
              "sig2" = sig2, "pi1" = pi1, "gamma" = gamma_ick)
  return(rtn)
}
```

In the above function, one of the most critical things are the starting values. Those were the most difficult to get, since the objective function is highly nonconcave.

What I essentially did was that I identified that there seems to be two clustered potentially partitioned by $y = 0$. Then I took subsets of the data determined by the rules $y < 0$ and $y \geq 0$, and fit individual linear regression models on that. The eventual βs I got were my respective starting values for β_1 and β_2, and the variance on the residuals from those were my starting values for the variances. Starting value of π_1 was just the proportion of y s less than 0. If we don't choose wise starting values we will get a very different fit (and not as good).

Then finally I will implement cross-validation to choose λ_1 and λ_2:

```
data <- read.csv("assign4_train.csv")
y <- data[, 1]
X <- as.matrix(data[, -1])
n <- length(y)
p <- dim(X)[2]

set.seed(1)
permutation <- sample(1:n, replace = FALSE)
K <- 5

# Making a list of indices for each split
test.index <- split(permutation, rep(1:K, length = n, each = n/K))

track.cv <- 0
lam1.vec <- seq(.01, 20, length = 10)
lam2.vec <- seq(.01, 20, length = 10)

track.cv <- matrix(0, nrow = length(lam1.vec), ncol = length(lam2.vec))
colnames(track.cv) = lam2.vec
rownames(track.cv) = lam1.vec
for(l1 in 1:length(lam1.vec))
{
```

```
  print(l1)
  lam1 <- lam1.vec[l1]
  for(l2 in 1:length(lam2.vec))
  {
    lam2 <- lam2.vec[l2]
    for(i in 1:K)
    {
      #print(i)
      train.ind <- test.index[[i]]
      # Making training data
      X.train <- X[-train.ind,] # removing ith X
      y.train <- y[-train.ind]  #removing ith y

      # fitting model for training data
      trained <- fitTrain(y = y.train, X = X.train, lam1 = lam1, lam2 = lam2)

      pred.y <- trained$pi1*X[train.ind, ] %*% trained$beta1 +
        (1 - trained$pi1)*X[train.ind, ] %*% trained$beta2

      # test error
      track.cv[l1, l2] <- track.cv[l1, l2] + sum((y[train.ind] - pred.y)^2)
    }
  }
}
```

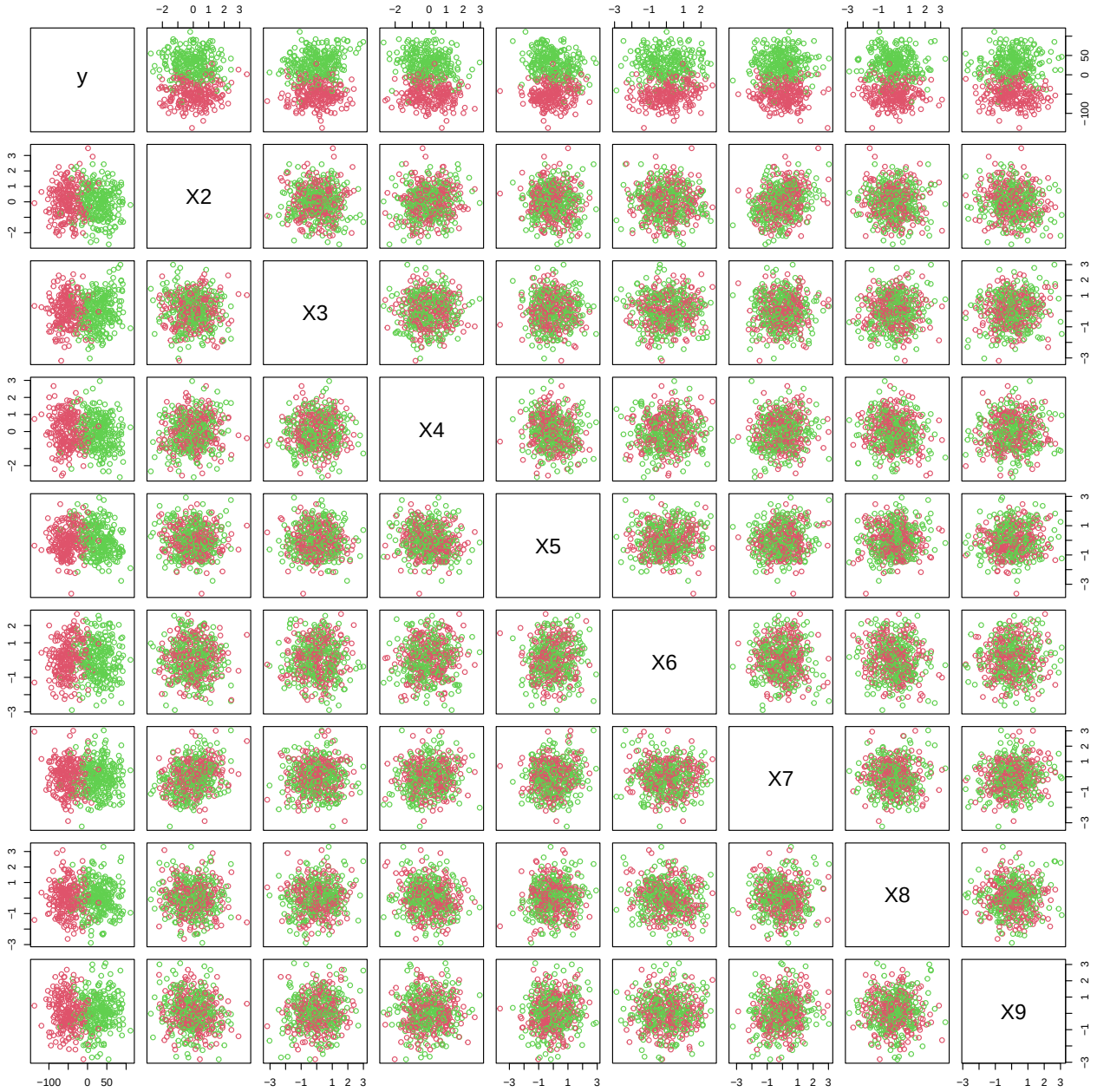
```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
CVerror <- track.cv/n

# chosen lam1 and lam2
temp <- which(CVerror == min(CVerror), arr.ind = TRUE)
chosen.lam1 <- lam1.vec[temp[1]]
chosen.lam2 <- lam2.vec[temp[2]]
```

```
final.fit  <- fitTrain(y = y, X = X, lam1 = chosen.lam1, lam2 = chosen.lam2)
save(final.fit, file = "MYfit_params.Rdata")
```

Let's analyse the fit again, to see what we actually get. Recall that conditional of each x_i, the y_i are either in class 1 or class 2. The `gamma_ick` store the probability of associations of each fit.

```
Zs <- apply(final.fit$gamma, 1, which.max)

plot(dat[ ,c(1,3:10)], col = Zs + 1)
```



Look how wonderfully the ys separate! So, beautiful, so elegant, just looking like a wow.

One issue

Notice that in the cross-validation part, I used the following code to obtain the predicted value of y.

```
pred.y <- trained$pi1*X[train.ind, ] %*% trained$beta1 +
        (1 - trained$pi1)*X[train.ind, ] %*% trained$beta2
```

This is because, having obtained my model fit, my most reasonable estimator of y can be obtained using

$$E[y_i|x_i] = p\, x_i^T \beta_1 + (1-p)\, x_i^T \beta_2 \,.$$

This is what my predict function uses.

For a real dataset this is an issue. Ideally, I would like to be able too choose which class y_i belongs to, based on the xs, and then pick the mean for that class. However, I can't do this, because the $\Pr(Z_i = 1) = p$ is independent of i! Ideally, each p should be p_i, where the class association probabilities can be modeled based on x_i and another $\delta \in \mathbb{R}^p$. I didn't generate the data like that because I thought that would be too much to expect y'all to fit! But please try this if you want.

How I generated the data

Below is the exact code I used to generate the data. You can use this to generate the testing data and see how different models perform.

```
library(expm)
set.seed(1)
n <- 1e3
p <- 100
sig1 <- 3
sig2 <- .01

## covariate matrix
rmat <- matrix(rnorm((p-1)^2), p-1, p-1)
cov_mat <- rmat%*%t(rmat)
corr_mat <- cov_mat/sqrt(diag(cov_mat)%*%t(diag(cov_mat)))
sq_cov <- sqrtm(corr_mat)

X <-  matrix(rnorm(n*(p-1) , sd = .1), nrow = n, ncol = p-1) %*% sq_cov
X <- scale(X)
X <- cbind(1, X)
```

```r
## regression coefficients
beta1 <- rep(0, p)
sub1 <- sample(1:p, size = p/10, replace = FALSE)
beta1[sub1] <- rnorm(p/10, mean = 10, sd = 3)
beta1[1] <- -50

beta2 <- rnorm(p, mean = 0, sd = .1)
sub2 <- sample(1:p, size = p/10, replace = FALSE)
beta2[sub2] <- rnorm(p/10, mean = -10, sd = 1)
beta2[1] <- 30


# latent variable
z <- rbinom(n = n, size = 1, prob = .40)
index <- z == 1

# generating response
y <- rep(0, n)
y[index] <- rnorm(sum(z), mean = (X %*% beta1)[index], sd = sig1)
y[!index] <- rnorm(n - sum(z), mean = (X %*% beta2)[!index], sd = sig2)

train.y <- head(y, n/2)
train.X <- head(X, n/2)
train <- data.frame(train.y, train.X)
colnames(train)[1] <- "y"
write.csv(train, file = "assign4_train.csv", row.names = FALSE)



test.y <- tail(y, n/2)
test.X <- tail(X, n/2)
test <- data.frame(test.y, test.X)
write.csv(test, file = "assign4_test.csv", row.names = FALSE)
```