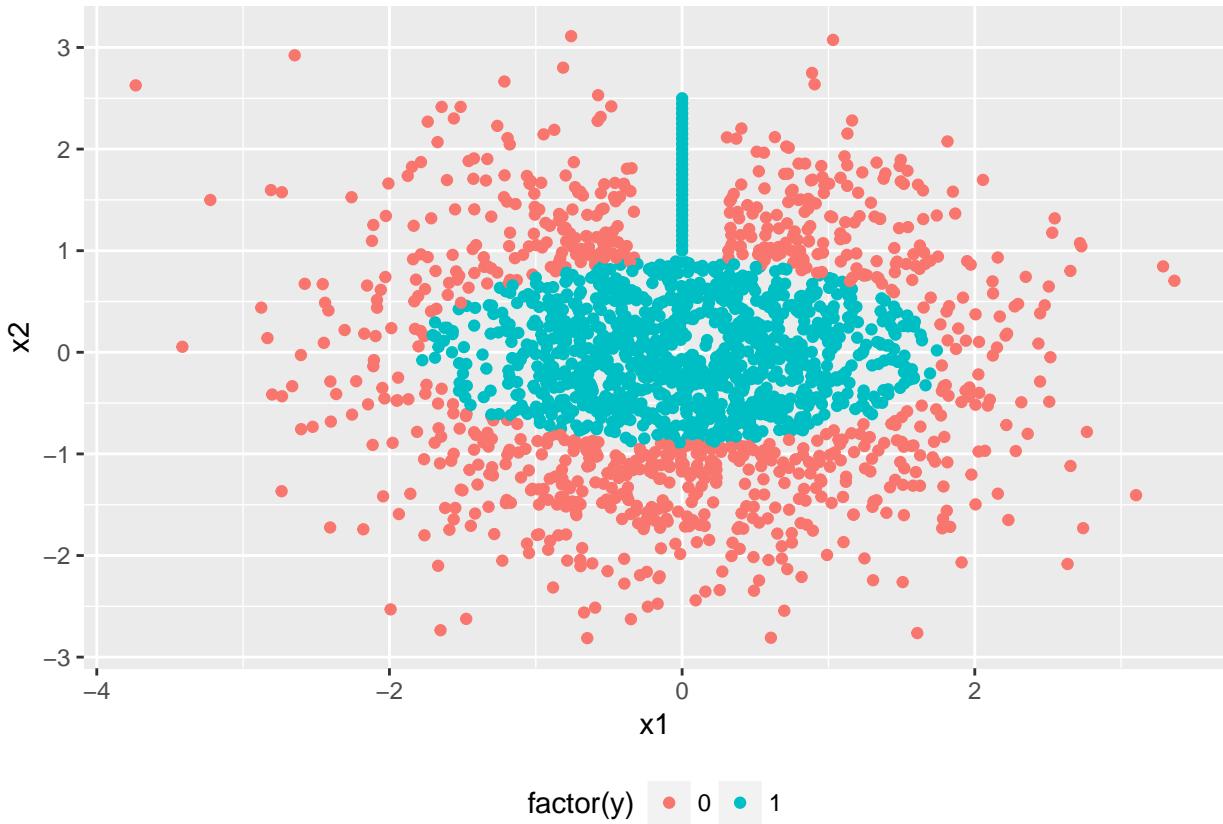


2. In this problem you will implement a neural network to solve a classification problem. To keep things simple - since time is limited - the data will consist of covariates $x^{(i)} \in \mathbb{R}^2$ and a response $y_i \in \{0, 1\}$ for $i = 1, 2, \dots, N$ (notice y takes on only two possible values). The classification problem involves fitting the model $y \sim f(x)$ over functions $f(x)$ that can be parameterized by our neural net, which is described below. The attached file `nn.txt` contains the samples. Each sample, corresponding to a row in the file, gives the three values $(x_1^{(i)}, x_2^{(i)}, y_i)$.

- (a) Visualize the dataset by plotting it with different colors for the two classes of y .

```
setwd("G:\\math\\504");require(ggplot2,quietly=T)
nn<-read.table("nn.txt",header=T)
p <- ggplot(nn, aes(x1, x2)) + geom_point(aes(colour = factor(y))) +
  theme(legend.position="bottom");p
```



- (b) Let η be the parameters of the neural net (i.e. all the α 's and β 's). What is the dimension of η in terms of m ?

$$\dim(\eta) = m \cdot (2 + 3) + 2$$

- (c) Write a function $NN(x, \eta, m)$ which takes a sample $x \in \mathbb{R}^2$ and a choice for η and returns the values of Y_1 and Y_2 . (Hint: It may be helpful to write functions such as `get_alpha(eta, i)`, which given η and i returns $\alpha^{(i)}$, and `get_alpha_0(eta, i)`, which given η and i returns $\alpha_0^{(i)}$. Using such functions will greatly simplify your code.)

```
X<-as.matrix(nn[,-3])

sigmoid<-function(a,w){ (1+exp(-a[1]-apply(w,1, function(w) t(a[-1])%*%w )) ) ) ^(-1)}

NN<-function(x,eta,m){

  X<-as.matrix(x)

  Alpha<-matrix(eta[1:prod( c(dim(X)[2] + 1, m) )], dim(X)[2] + 1 , m)
  Z<-apply(Alpha,2, function(v) sigmoid(v,X))

  Beta<-matrix( eta[(prod(dim(Alpha))+1):length(eta) ] , dim(Z)[2]+1, 2)

  T<-apply(Beta,2, function(v) sigmoid(v,Z))
  Y<- apply(T,2, function(x) exp(x))
  Y<-Y/rowSums(Y)
  return(Y)
}

#test
head( NN(nn[,-3], runif(22,-1,1) , 4) )

##          [,1]      [,2]
## [1,] 0.5046086 0.4953914
## [2,] 0.4895462 0.5104538
## [3,] 0.5159363 0.4840637
## [4,] 0.4898929 0.5101071
## [5,] 0.4943930 0.5056070
## [6,] 0.5127321 0.4872679
```

- (d) Explain why the log likelihood function $\log L(\eta)$ for the neural net is given by

$$\log L(\eta) = \sum_{i=1}^N (1 - y_i) \log(Y_1) + y_i \log(Y_2). \quad (1)$$

(In class, when I wrote the log likelihood, I forgot the log on the Y_1 and Y_2 !) Write a function that computes $\log L(\eta)$ (you will need to pass the data to the function). Write a function that uses finite difference to compute the gradient of $\log L(\eta)$.

We are trying to classify each data point by a function that approximates $P(Y|X)$. This is similar to logistic (or other binary regression) because we have two classes and we can derive the likelihood in the same way. Let $\pi = P(y = 1|x)$.

$$\begin{aligned} \mathcal{L}(\eta) &= \prod_{i=1}^N \pi_i^{y_i} (1 - \pi_i)^{1-y_i} \\ \ell(\eta) &= \sum_{i=1}^N y_i \log(\pi_i) + (1 - y_i) \log(1 - \pi_i). \end{aligned}$$

For our neural net, we know our $Y_1 \sim P(y = 0|x)$ and $Y_2 \sim P(y = 1|x)$, so:

$$\begin{aligned}\ell(\eta) &= \sum_{i=1}^N (1 - y_i) \log(1 - \pi_i) + y_i \log(\pi_i) \\ &= \sum_{i=1}^N (1 - y_i) \log(Y_1) + y_i \log(Y_2).\end{aligned}$$

We look at our i th data sample and it can be in one of two classes, $y = 0$ or $y = 1$. If the sample is of class 0 and the prediction for its probability is Y_1 , and if it is of class 1 the prediction for its probability is Y_2 . In the case of neural nets Y_1 and Y_2 are nested functions that depend on η and x_i .

```
gradL<-function(eta,x,y,m ){
  y<-as.matrix(y)
  dimeta<-m*(dim(x)[2]+1)+2*(1+m)
  gradf<-rep(NA,dimeta)

  Y<-NN(x,eta,m)
  ETA<-eta

  logl<-sum( (1-y)*log(Y[,1]) + ( y)*log(Y[,2]) )
  for( i in 1:dimeta){

    ETA[i]<-eta[i]+(10^-6)

    Yp<-NN(x,ETA,m)

    #finite difference
    gradf[i]<-(sum( (1-y)*log(Yp[,1]) + ( y)*log(Yp[,2]) ) - logl ) / (10^-6) }

  return(gradf)
}

#test
gradL(runif(22,-1,1),nn[,-3],nn[,3],4)

## [1] -0.8283328 -0.9883556 -0.7351650 14.5786594 14.3632383
## [6] 18.5654185 7.7527188 6.8544475 -0.8241443 2.5202667
## [11] 2.2725317 3.8197443 -31.1889273 -48.4908521 -61.7618916
## [16] -76.5041611 -88.6678185 -55.7797850 -39.6477753 -28.3507788
## [21] -14.0341956 -3.3852555
```

- (e) Set $m = 4$ and train your neural net by maximizing the $\log L(\eta)$ using steepest ascent. (It took me roughly 45 minutes of run time to get a good fit, roughly 3000 iterations. Your results may vary from this depending on implementation and hardware.)

```
logL<-function(x,y,eta,m){
  Y<-NN(x,eta,m)
  return( sum( (1-y)*log(Y[,1]) + ( y)*log(Y[,2]) ) ) }
```

```
j=1;current_grad <- gradL(Alpha , X , nn[,3] , 4 )

for(i in j:(3000)) {

  d <- current_grad/Norm(current_grad)

  s <- 1
  current_logL <- logL(X,nn[,3],Alpha,4)

  while(logL(X,nn[,3],Alpha+s*d,4) < current_logL)
    s <- s/2

  Alpha <- Alpha + s*d
  current_grad <- gradL(Alpha , X , nn[,3] , 4 ) }
```

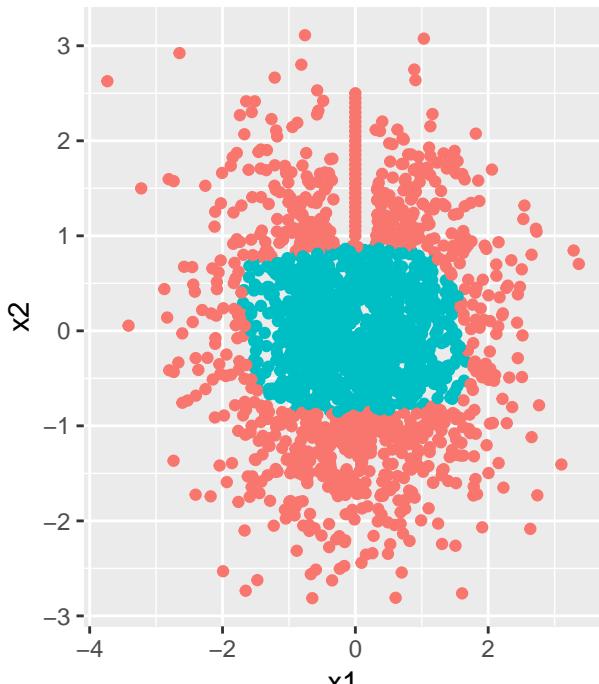
#Alpha is eta (alphas and betas)

Alpha

```
## [1] -3.7748365  2.1178734  0.3139542 -5.5415006 -0.5958018
## [6]  5.8168459 11.0561352  0.2074085 14.0207743 25.2868297
## [11] 16.0428078  3.8300725 15.9792114 23.4815794 23.2210775
## [16] -13.5097426 -12.2073593 -18.6510642 -27.0546378 -27.4979835
## [21] 15.2350215 14.7081561
```

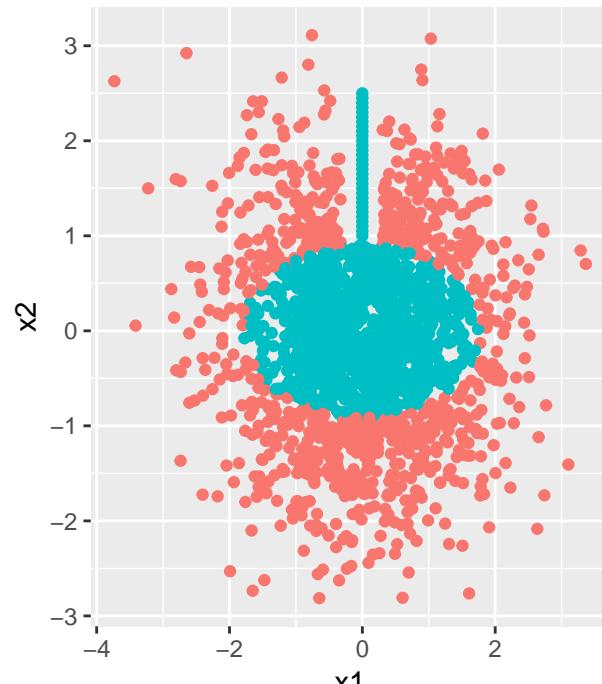
```
multiplot(ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y3))) +
  theme(legend.position="bottom") + ggtitle("Test") , p + ggtitle("Actual") , cols=2 )
```

Test



factor(y3) ● 0 ● 1

Actual



factor(y) ● 0 ● 1

Decent fit with a cutoff of $p = 0.5$, but it is not overfitting the “tail” part.

- (f) Remember that a classifier in this case is a function $F(x) : \mathbb{R}^2 \rightarrow \{0, 1\}$, where $x \in \mathbb{R}^2$. Once you choose η by computing the maximum likelihood in (e), choose a cutoff $p \in [0, 1]$. Set F by

$$F(x) = \begin{cases} 0 & \text{if } Y_1(x, \eta) < p \\ 1 & \text{if } Y_1(x, \eta) \geq p \end{cases} \quad (2)$$

Try different value of p and for each p , visualize your classifier.

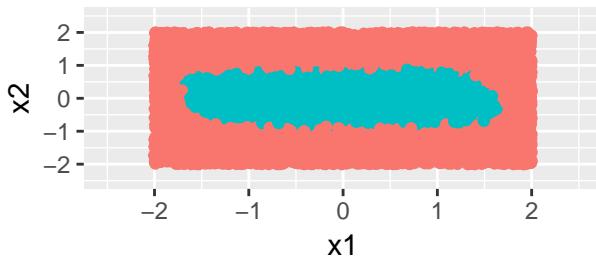
```

x1<-matrix(4*runif(20000 )-2, 10000,2)
test<-as.data.frame((NN(x1,Alpha ,4)))
test<-cbind(test,x1)
test$y1<- ifelse( test[,1]>.5 , 0,1)
test$y2<- ifelse( test[,1]>.7 , 0,1)
test$y3<- ifelse( test[,1]>.3 , 0,1)
names(test)[3:4]<-c("x1", "x2")

p1 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y1))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.5") +
  theme(legend.position="bottom")
p2 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y2))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.7") +
  theme(legend.position="bottom")
p3 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y3))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.3") +
  theme(legend.position="bottom")
p<- p + ggtitle("original data")
multiplot(p1, p2, p3,p, cols=2)

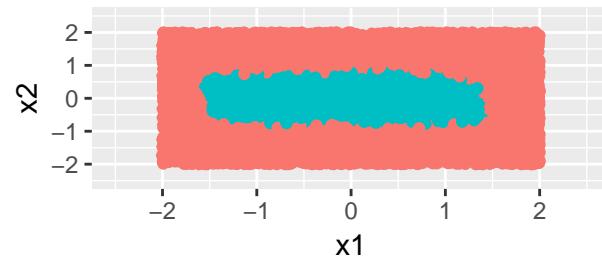
```

$p = 0.5$



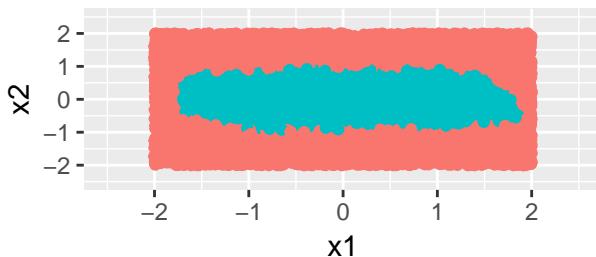
factor(y1) ● 0 ● 1

$p = 0.3$



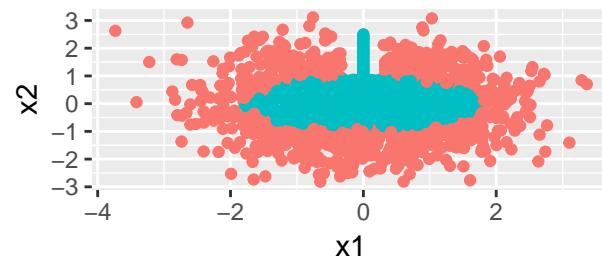
factor(y3) ● 0 ● 1

$p = 0.7$



factor(y2) ● 0 ● 1

original data



factor(y) ● 0 ● 1

- (g) Now repeat (e) and (f), but in your log likelihood, include a penalty term of the form,

$$\rho \left(\sum_{i=1}^m \|\alpha^{(i)}\|^2 + \sum_{j=1}^2 \|\beta^{(j)}\|^2 \right). \quad (3)$$

Given that we are maximizing, explain why you should subtract the penalty term rather than add it onto the log-likelihood in (d). Train your net with several values of ρ , visualize the resulting classifier for different p , and compare your results.

We wish to maximize the likelihood (or alternatively the log-likelihood) so in order to impose a penalty we will reduce the likelihood. This is done by subtracting the penalty term rather than adding it.

```
#Penalized Log likelihood function
PlogL<-function(x,y,eta,m,rho){
  X<-as.matrix(x)
  ALPHA<-matrix(eta[1:prod(c(dim(X)[2] + 1, m))], dim(X)[2] + 1 , m)
  Beta<-matrix(eta[(prod(c(dim(X)[2] + 1, m))+1):length(eta)],
    m+1,2)
  ALPHA<-as.vector(t(ALPHA[-1 ,])) ;Beta<-as.vector(t(Beta[-1 ,]))
  Norm <- function(w){ sqrt(sum(w^2))}

  Y<-NN(x,eta,m)
  return( sum( (1-y)*log(Y[,1]) + ( y)*log(Y[,2]) ) -
    rho*(sum(Norm(ALPHA)^2)+sum(Norm(Beta)^2) )
  ) }
```

$$\rho = 0.1$$

```
j=1;current_grad <- gradL(Alpha , X , nn[,3] , 4 )
system.time(
  for(i in j:(2000)) {
    #iter <- iter + 1

    d <- current_grad/Norm(current_grad)

    s <- 1
    #Penalized Log likelihood
    current_logL <- PlogL(X,nn[,3],Alpha,4,.1)

    while(logL(X,nn[,3],Alpha+s*d,4) < current_logL)
      s <- s/2

    Alpha <- Alpha + s*d
    current_grad <- gradL(Alpha , X , nn[,3] , 4 ) } )

j=i

## [1] -4.217173 2.129696 2.291096 -5.749465 -1.505578 5.422963
```

```

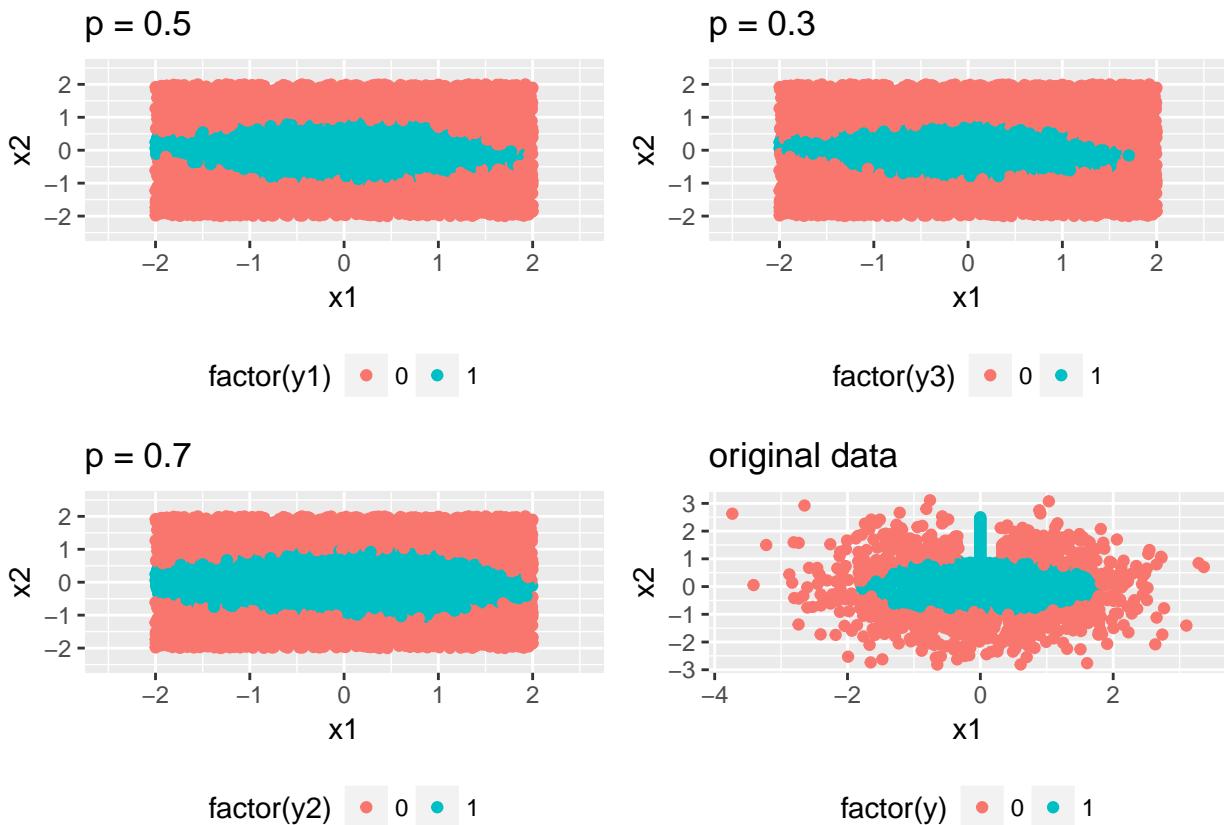
## [7] 10.917247 -2.730573 12.886107 10.326642 5.094741 9.110171
## [13] 13.503592 21.919573 22.305991 -10.991376 -10.851156 -14.688772
## [19] -23.092542 -24.351591 12.003044 12.654236

x1<-matrix(4*runif(10000 )-2, 5000,2)
test<-as.data.frame((NN(x1,Alpha ,4)))
test<-cbind(test,x1)
test$y1<- ifelse( test[,1]>.5 , 0,1)
test$y2<- ifelse( test[,1]>.7 , 0,1)
test$y3<- ifelse( test[,1]>.3 , 0,1)
names(test)[3:4]<-c("x1","x2")

p1 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y1))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.5") +
  theme(legend.position="bottom")
p2 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y2))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.7") +
  theme(legend.position="bottom")
p3 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y3))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.3") +
  theme(legend.position="bottom")

multiplot(p1, p2, p3, p,cols=2)

```



```

j=1;current_grad <- gradL(Alpha , X , nn[,3] , 4 )
system.time(
for(i in j:(2000)) {
  #iter <- iter + 1

  d <- current_grad/Norm(current_grad)

  s <- 1
  current_logL <- PlogL(X,nn[,3],Alpha,4,1)

  while(logL(X,nn[,3],Alpha+s*d,4) < current_logL)
    s <- s/2

  Alpha <- Alpha + s*d
  current_grad <- gradL(Alpha , X , nn[,3] , 4 ) } )

j=i

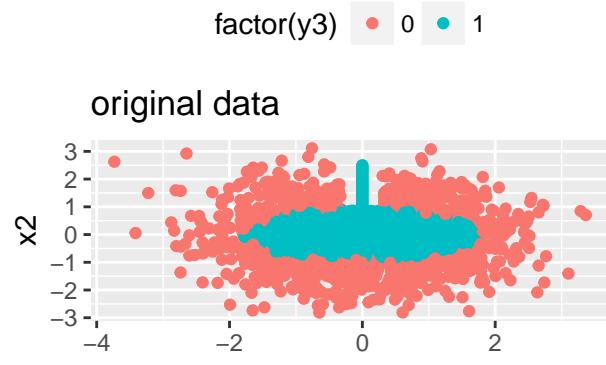
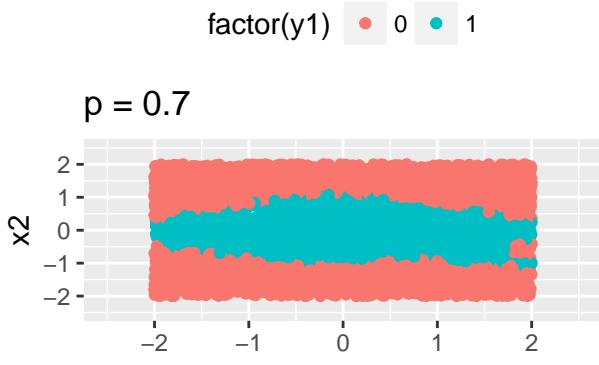
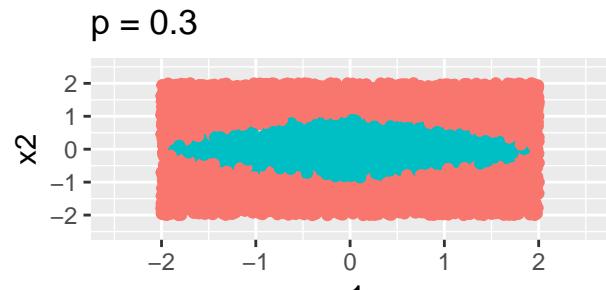
## [1] -5.012726  2.095720  3.540725 -5.324588 -2.108554  3.957979
## [7]  9.760088 -4.196175  9.938802  6.688340  2.643233  7.210336
## [13] 15.743670 24.279396 24.855088 -12.204780 -14.039753 -18.199320
## [19] -26.719915 -28.098740 13.621660 15.881839

x1<-matrix(4*runif(10000 )-2, 5000,2)
test<-as.data.frame((NN(x1,Alpha ,4)))
test<-cbind(test,x1)
test$y1<- ifelse( test[,1]>.5 , 0,1)
test$y2<- ifelse( test[,1]>.7 , 0,1)
test$y3<- ifelse( test[,1]>.3 , 0,1)
names(test)[3:4]<-c("x1","x2")

p1 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y1))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.5") +
  theme(legend.position="bottom")
p2 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y2))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.7") +
  theme(legend.position="bottom")
p3 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y3))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.3") +
  theme(legend.position="bottom")

multiplot(p1, p2, p3, p, cols=2)

```



$$\rho = 100$$

```
j=1;current_grad <- gradL(Alpha , X , nn[,3] , 4 )
system.time(
  for(i in j:(2000)) {

    d <- current_grad/Norm(current_grad)

    s <- 1
    current_logL <- PlogL(X,nn[,3],Alpha,4,100)

    while(logL(X,nn[,3],Alpha+s*d,4) < current_logL)
      s <- s/2

    Alpha <- Alpha + s*d
    current_grad <- gradL(Alpha , X , nn[,3] , 4 ) } )

j=i
```

```
## [1] -4.80870303  2.51425207 -0.70430171 -8.39809328  0.07027287
## [6]  8.89975250 13.39129423  0.16705617 16.70429985 19.56583078
## [11] 10.80716806 10.69527465 12.34268115 20.81992539 21.16594983
## [16] -9.72328246 -10.27706135 -12.66218131 -21.14339824 -22.67591064
## [21]  9.49389847 10.38622987
```

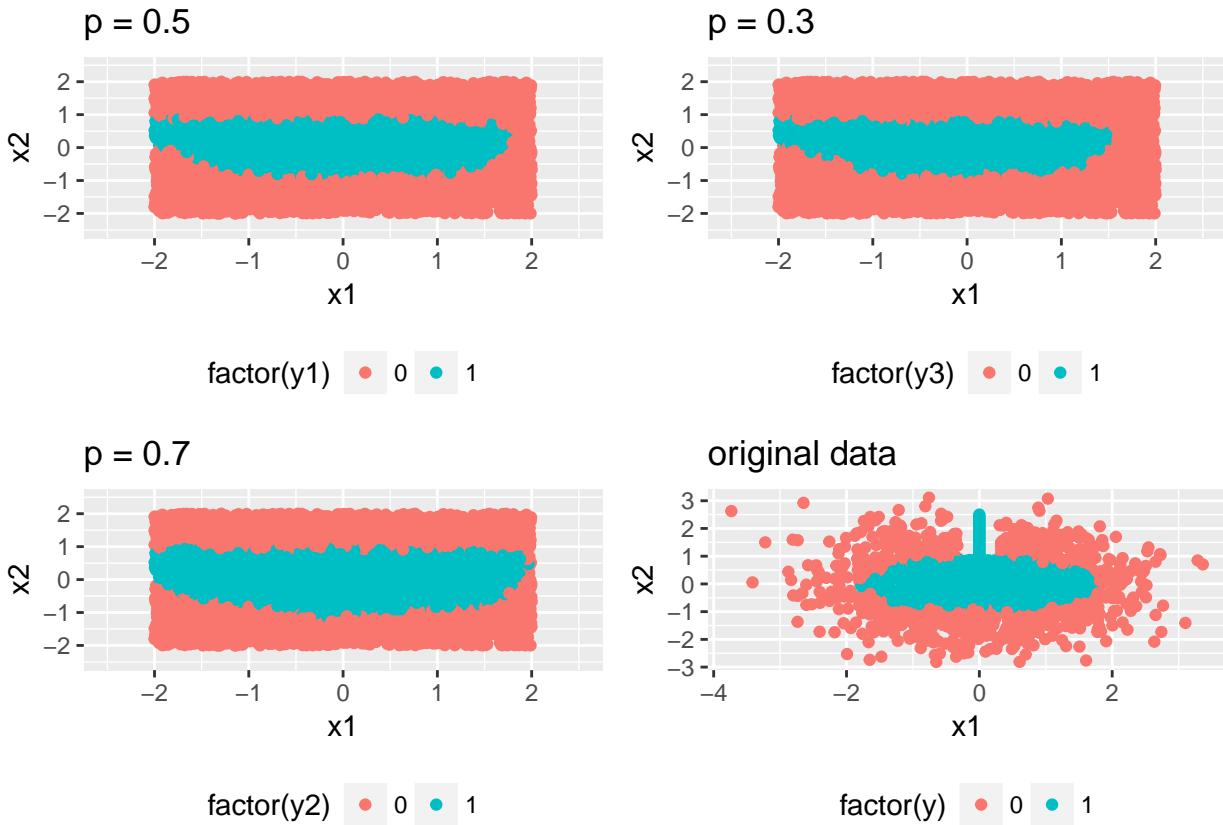
```

x1<-matrix(4*runif(10000 )-2, 5000,2)
test<-as.data.frame((NN(x1,Alpha ,4)))
test<-cbind(test,x1)
test$y1<- ifelse( test[,1]>.5 , 0,1)
test$y2<- ifelse( test[,1]>.7 , 0,1)
test$y3<- ifelse( test[,1]>.3 , 0,1)
names(test)[3:4]<-c("x1","x2")

p1 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y1))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.5") +
  theme(legend.position="bottom")
p2 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y2))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.7") +
  theme(legend.position="bottom")
p3 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y3))) +
  ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.3") +
  theme(legend.position="bottom")

multiplot(p1, p2, p3, p,   cols=2)

```



The non-penalized data was a decent fit, but was almost rectangular in nature. The penalized likelihood example data are giving it more of a flatter, oval shaped fit, with specific good fits with moderate cutoffs at $p = 0.5$ and smaller ρ 's at 0.1 and 1. It looks like the heavily penalized data, $\rho = 100$, is getting a bit of the shape chopped off, especially at the top.

3. Repeat problem 2, but now use a logistic regression to build your classifier. How does the logistic regression classifier compare to the neural net classifier?

```

nn.fit<-glm(y~,data = nn, family = "binomial")
logclass<-cbind(nn,predict(nn.fit,type="response"))

logclass$yhat<-ifelse( logclass[,4]> .5 , 0,1)
p1 <- ggplot(logclass, aes(x1, x2)) + geom_point(aes(colour = factor(yhat))) +
  stat_function(fun=function(x){ -(0.2447 /0.2031 ) -(0.0186*x/0.2031 ) -
    (log((1/.5)-1) /0.2031 ) }, geom="line" , color = "black") + ggtitle("p = 0.5")+
  theme(legend.position="bottom")

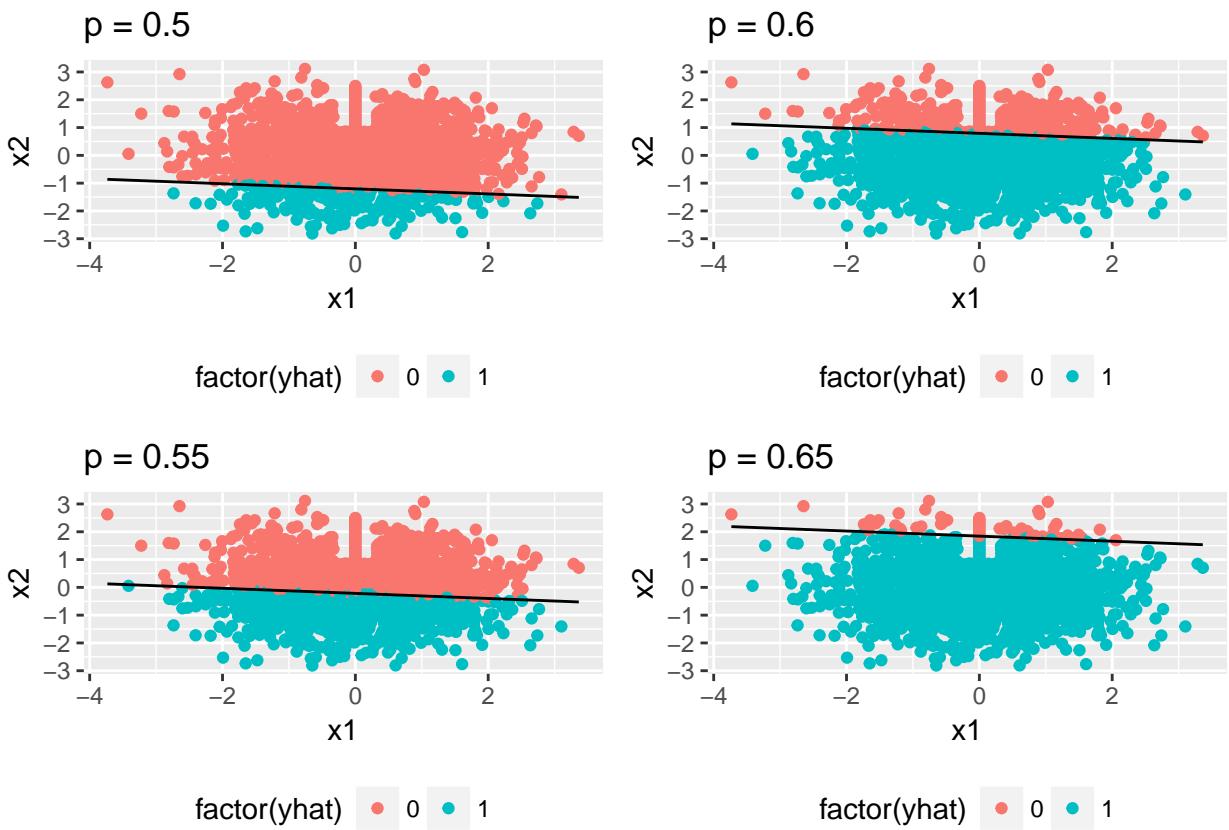
logclass$yhat<-ifelse( logclass[,4]> .55 , 0,1)
p2 <- ggplot(logclass, aes(x1, x2)) + geom_point(aes(colour = factor(yhat))) +
  stat_function(fun=function(x){ -(0.2447 /0.2031 ) -(0.0186*x/0.2031 ) -
    (log((1/.55)-1) /0.2031 ) }, geom="line" , color = "black") + ggtitle("p = 0.55")+
  theme(legend.position="bottom")

logclass$yhat<-ifelse( logclass[,4]> .6 , 0,1)
p3<- ggplot(logclass, aes(x1, x2)) + geom_point(aes(colour = factor(yhat))) +
  stat_function(fun=function(x){ -(0.2447 /0.2031 ) -(0.0186*x/0.2031 ) -
    (log((1/.6)-1) /0.2031 ) }, geom="line" , color = "black") + ggtitle("p = 0.6")+
  theme(legend.position="bottom")

logclass$yhat<-ifelse( logclass[,4]> .65 , 0,1)
p4 <- ggplot(logclass, aes(x1, x2)) + geom_point(aes(colour = factor(yhat))) +
  stat_function(fun=function(x){ -(0.2447 /0.2031 ) -(0.0186*x/0.2031 ) -
    (log((1/.65)-1) /0.2031 ) }, geom="line" , color = "black") + ggtitle("p = 0.65")+
  theme(legend.position="bottom")

multiplot(p1, p2, p3, p4,  cols=2)

```



The logistic regression classifier is downright bad compared to the neural net. Since the linear predictor function can only construct a line or plane it is having a lot of trouble with our oval/ellipse shaped data. Our neural net classifier, both penalized and non-penalized, is doing a relatively decent job with both our training data and test data. The oval/ellipse is not giving it much trouble.