Michael Leibert

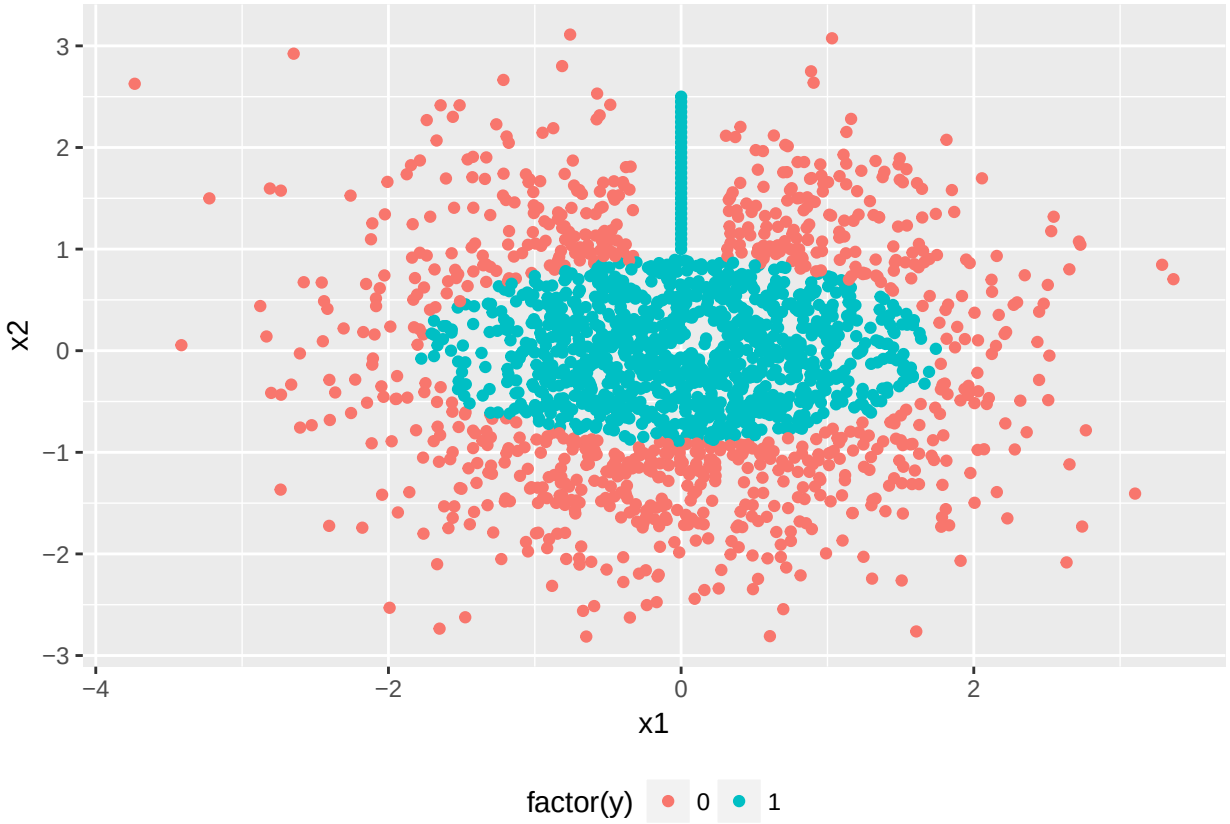
Math 504

Homework 12

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 $i = 1, 2, \ldots, N$ (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.)

```
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)
}

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

```
##           [,1]      [,2]
## [1,] 0.5999959 0.4000041
## [2,] 0.5432917 0.4567083
## [3,] 0.6077322 0.3922678
## [4,] 0.5991071 0.4008929
## [5,] 0.5917925 0.4082075
## [6,] 0.5947735 0.4052265
```

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

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

(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 $\log L(\eta)$.

EXPLAIN

```
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)

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

    return(gradf)
}

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

```
##  [1]  -15.57735  -17.46491  -20.56678  -19.67044  -19.29310  -19.92448
##  [7]  -38.82923  -42.42010  -44.92006  -51.35412  -52.39803  -52.91170
## [13] -106.33592 -128.62834 -159.76772 -187.48513 -211.44681 -159.03051
## [19] -135.86846 -106.30940  -78.29720  -54.03919
```

(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 implemenation 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
```

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

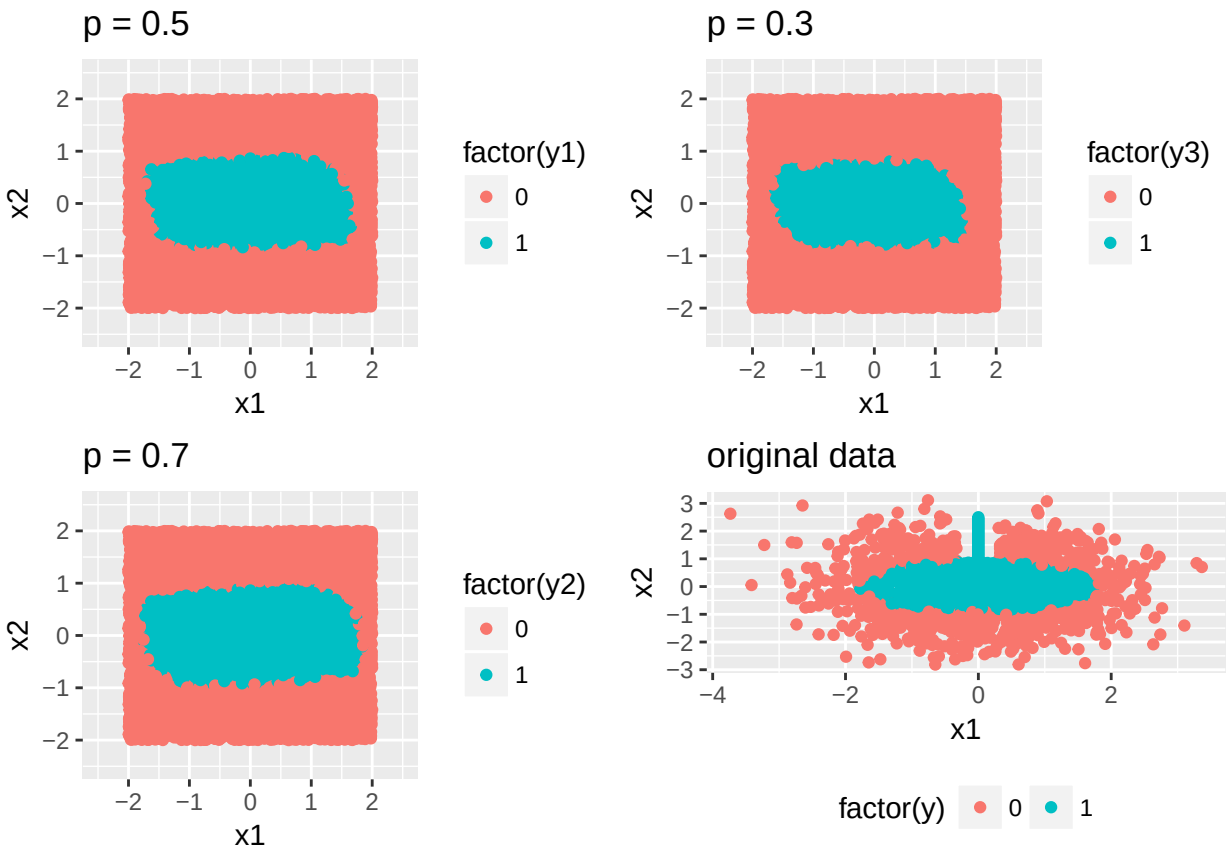
$$F(x) = \begin{cases} 0 & \text{if } Y_1(x, \eta) < p \\ 1 & \text{if } Y_1(x, \eta) \geq p \end{cases} \tag{2}$$

Try different value of p and for each p, visualize your classifier. You can do this in any way you like, but here is one way. Generate many points in $\mathbb{R}^2$, determine the predicted class of each using your classifier, and then plot as in part (a). You can generate random coordinates within $[-2, 2]$, which is roughly where all the data points lie, using

```
x1 <- 4*runif(10000) - 2
x2 <- 4*runif(10000) - 2
```

`runif(10000)` generates 10000 numbers uniformly distributed on $[0, 1]$.

```
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")
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")
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")
p<- p + ggtitle("original data")
multiplot(p1, p2, p3,p,  cols=2)
```



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

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

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 differnt p, and compare your results.

EXPLAIN

```
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 ,]))

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

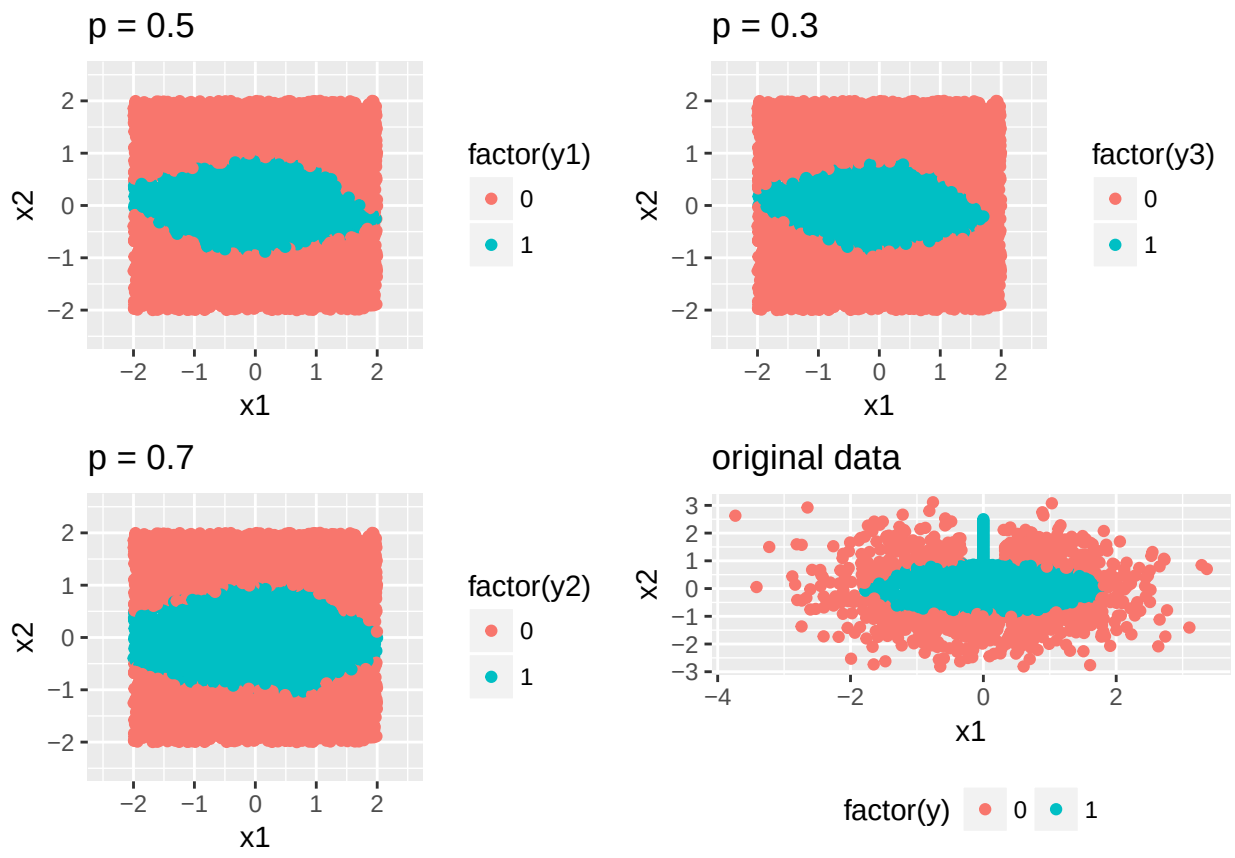
```
    return( sum( (1-y)*log(Y[,1]) + ( y)*log(Y[,2])  ) -
        rho*(sum(Norm(ALPHA)^2)+sum(Norm(Beta)^2) )
    )   }
```

$\rho = 0.1$

```
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
    #current_logL <- logL(X,nn[,3],Alpha,4)
    current_logL <- PlogL(X,nn[,3],Alpha,4,100)

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

    # update
    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")
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")
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")

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

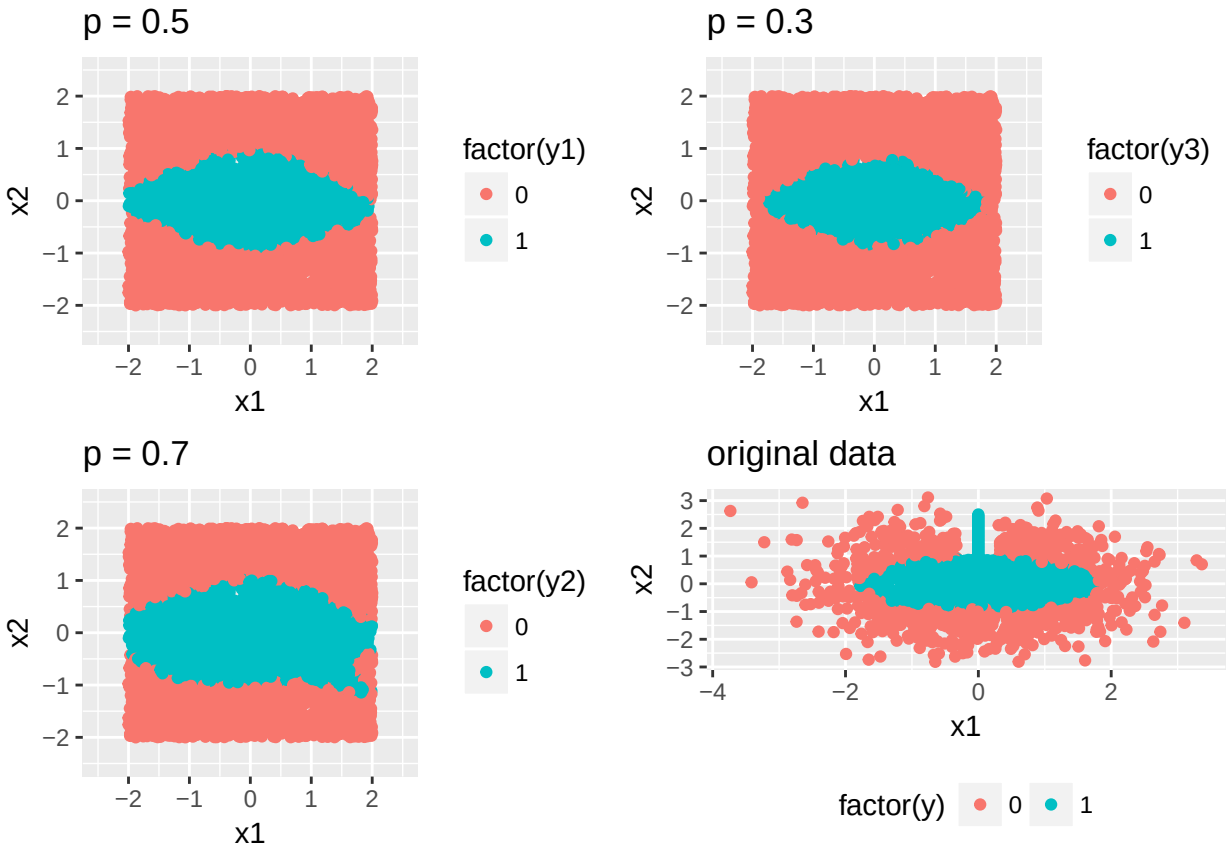


$\rho = 1$

```
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
    #current_logL <- logL(X,nn[,3],Alpha,4)
    current_logL <- PlogL(X,nn[,3],Alpha,4,1)

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

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


j=i
```

```
##  [1]  -4.449694   2.001816   3.202079  -5.040372  -1.882563   4.017746
##  [7]  10.261748  -3.801292  10.604388   6.988823   3.020531   7.565197
## [13]  13.557587  22.068377  22.603324 -10.790467 -11.260033 -15.279049
## [19] -23.705467 -25.003207  12.385907  13.578146
```

```
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")
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")
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")

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



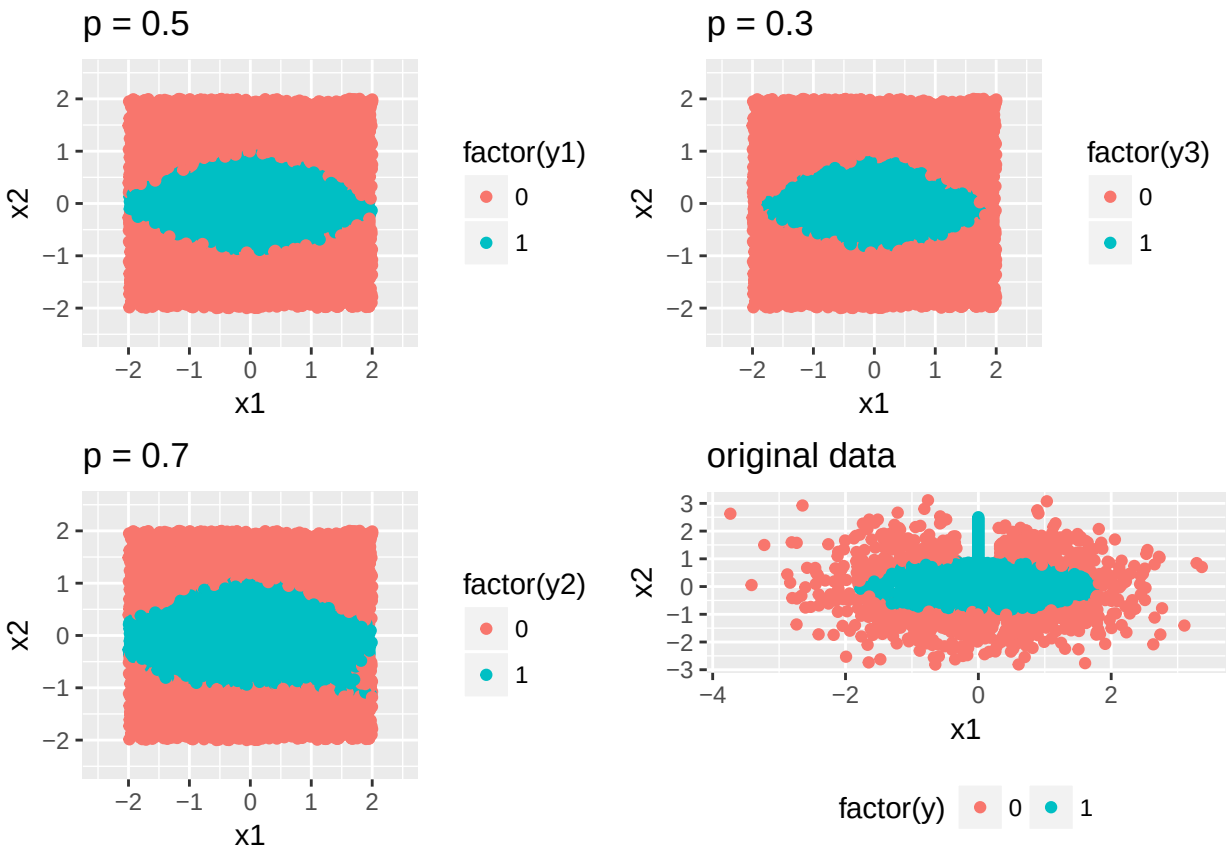
$\rho = 100$

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

```
 for(i in j:(2000)) {
    #iter <- iter + 1

    d <- current_grad/Norm(current_grad)

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

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

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


j=i
```

```
##  [1]  -4.449694   2.001816   3.202079  -5.040372  -1.882563   4.017746
##  [7]  10.261748  -3.801292  10.604388   6.988823   3.020531   7.565197
## [13]  13.557587  22.068377  22.603324 -10.790467 -11.260033 -15.279049
## [19] -23.705467 -25.003207  12.385907  13.578146
```

```
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")
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")
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")

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



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")

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")

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")

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")

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