Trabajo 1 Aprendizaje Automático

Antonio Mannuel Milán Jiménez

22 de marzo de 2017

Ejercicio 1

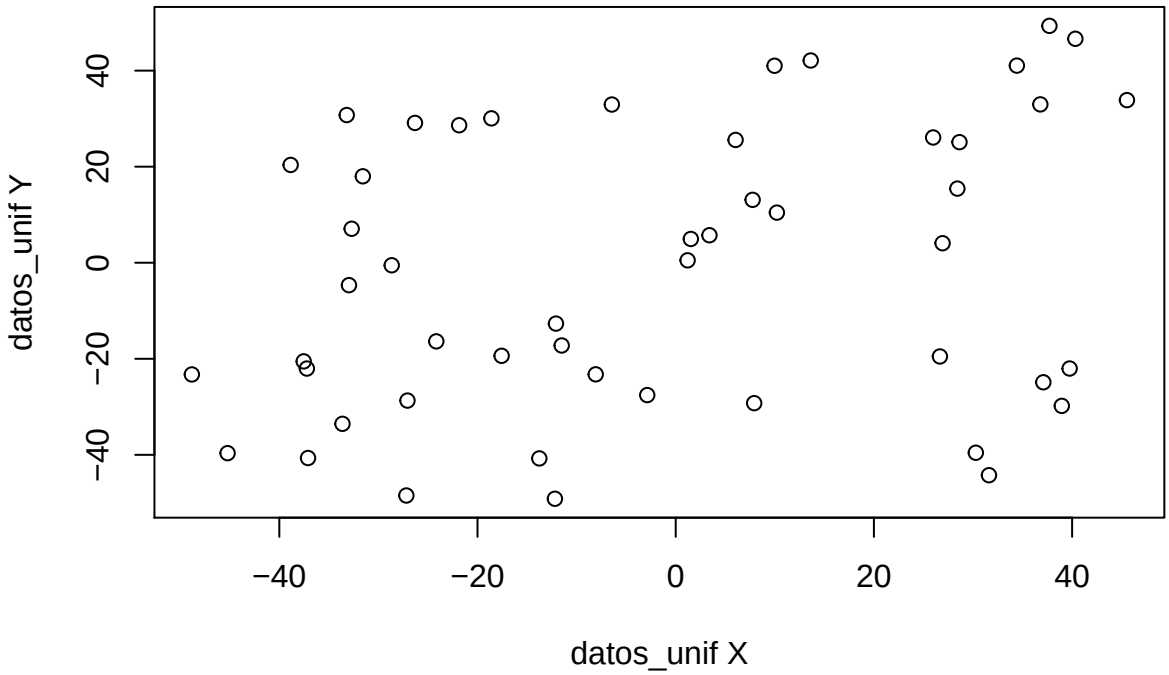
Se han hecho uso de las funciones **simula_unif**, **simula_gaus** y **simula_recta** proporcionadas en Decsai.

Apartado 1

Nube de puntos con **simula_unif** para N=50, dim=2 y rango = [-50,50].

La llamada a la función es:

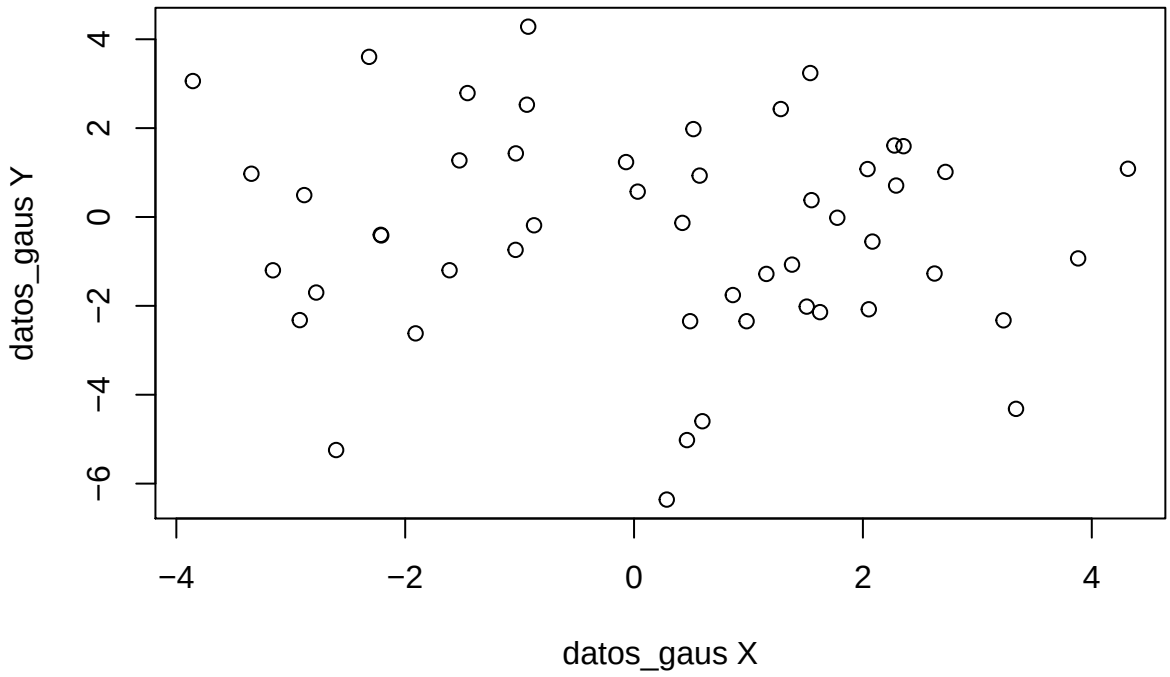
```
datos_unif=simula_unif(50,2,c(-50,50))
```



Nube de puntos con **simula_gaus** para N=50, dim=2 y sigma=[5,7]

La llamada a la función es:

```
datos_gaus=simula_gaus(50,2,c(5,7))
```



Apartado 2A

Hemos creado una función **generarEtiquetasMatriz** con la que a partir de unos datos y una recta, asigna a cada punto una etiqueta en función del signo respecto a la recta mediante f(x,y) = y-ax-b

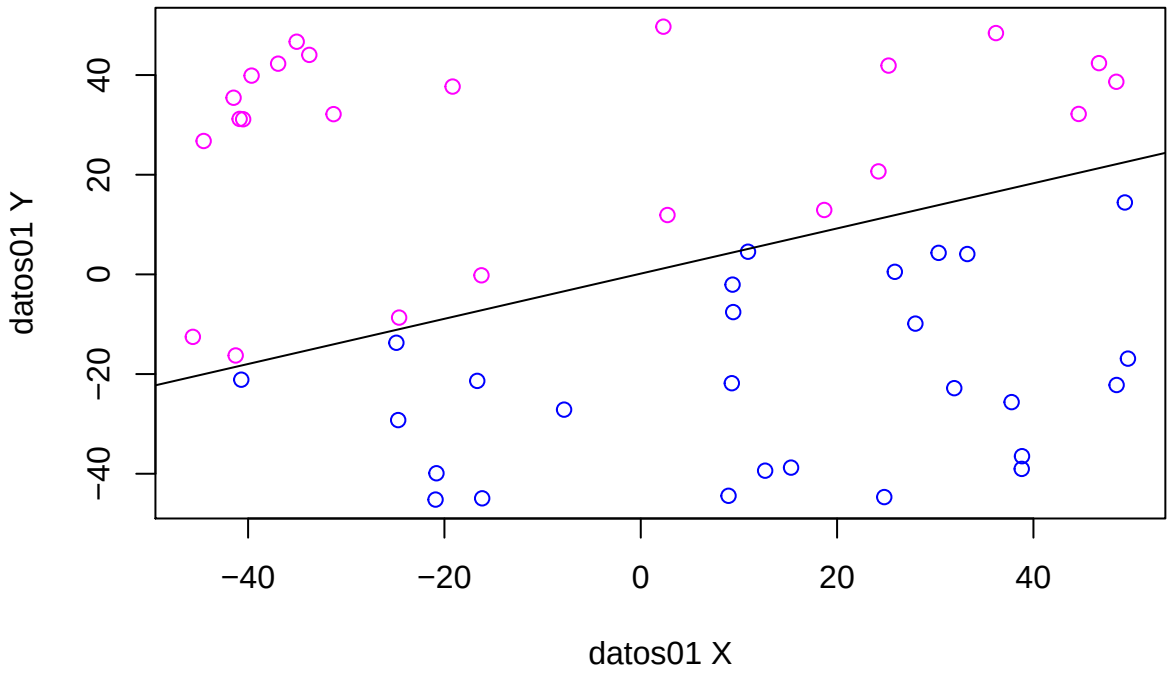
```
generarEtiqueta<-function(datos,a,b){
  sign(datos[2]-a*datos[1]-b)
}

generarEtiquetasMatriz<-function(matriz,a,b){
  apply(matriz,1,generarEtiqueta,a,b)
}
```

La llamada a la funcíón es:

etiquetas=generarEtiquetasMatriz(datos01,recta[1],recta[2])

Dibujando la nube de puntos con las etiquetas obtenidas y la recta que las ha determinado obtenemos la gráfica siguiente:



Apartado 2B

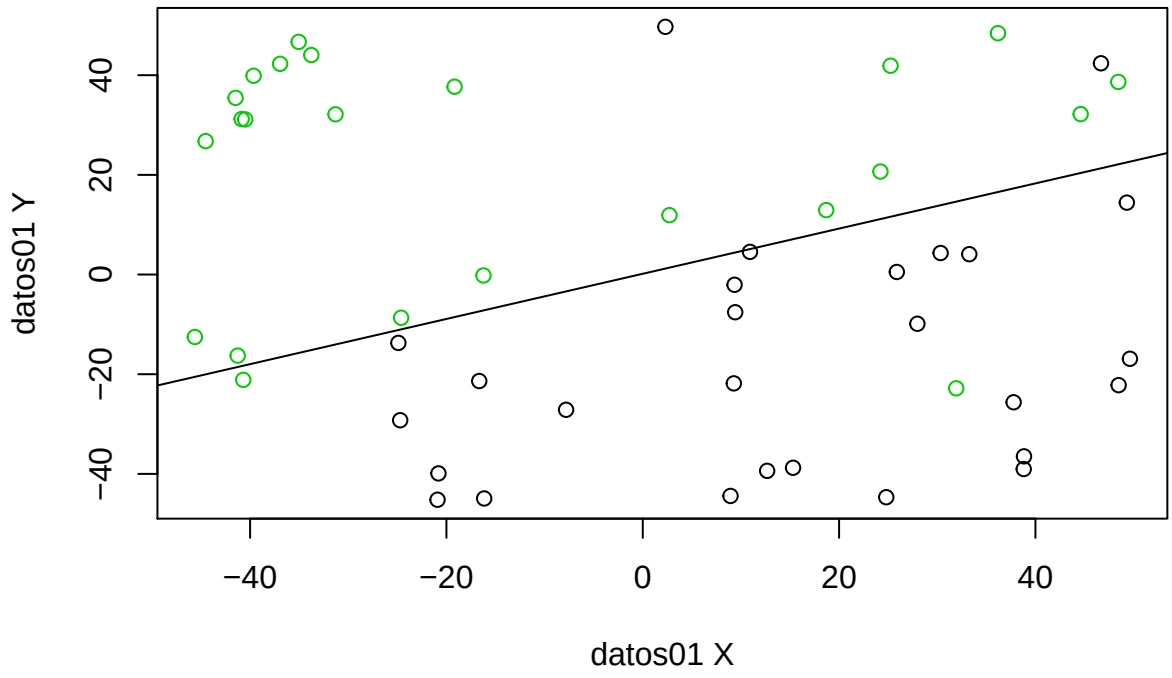
Para crear ruido, hemos creado la función **modificarEtiquetas** que altera en un porcentaje las etiquetas positivas a negativas y viceversa. Dado que separamos en un inicio las etiquetas positivas y negativas en dos conjuntos, hace que la misma etiqueta que se ha hecho positiva no pueda hacerse de nuevo negativa:

```
modificarEtiquetas<-function(etiquetas,porcentaje){
  etiqP = c()
  etiqN = c()
  n = length(etiquetas)
  for(i in 1:n){
    if(etiquetas[i]==1)
      etiqP = c(etiqP,i)
    else
      etiqN = c(etiqN,i)
  }
  etiqP = sample(etiqP)
  etiqN = sample(etiqN)
  nP = length(etiqP)*porcentaje/100
  nN = length(etiqN)*porcentaje/100
  for(i in 1:nP){
    t=etiqP[i]
    etiquetas[t] = -1
  }
  for(i in 1:nN){
    t=etiqN[i]
    etiquetas[t] = 1
  }
  etiquetas
}
```

La llamada a la función sería:

etiquetas_ruido=modificarEtiquetas(etiquetas,10)

Y aquí tenemos la gráfica del apartado anterior al incluirle ruido:



Apartado 3

Para este apartado hemos utilizado la función **pintar_frontera** modficado para que nos dibuje también la nube de puntos etiquetada. También hemos creado las 4 funciones que se nos pedían:

```
f1<-function(x,y){
  ((x-10)^2 + (y-20)^2 -400)
}
f2<-function(x,y){
  (0.5*(x+10)^2 + (y-20)^2 -400)
}
f3<-function(x,y){
  (0.5*(x-10)^2 - (y+20)^2 -400)
}
f4<-function(x,y){
  (y-20*x^2-5*x+3)
}

pintar_frontera = function(f,datos,etiquetas_ruido,rango=c(-50,50)) {
  x=y=seq(rango[1],rango[2],length.out = 100)
  z = outer(x,y,FUN=f)
  if (dev.cur()==1) # no esta abierto el dispositivo lo abre con plot
    plot(1, type="n", xlim=rango, ylim=rango)
  contour(x,y,z, levels = 0, drawlabels = FALSE,xlim =rango, ylim=rango, xlab = "x", ylab = "y")
  points(datos[,1],datos[,2],col=etiquetas_ruido+5)
}
```

Entonces la llamada a la función sería:

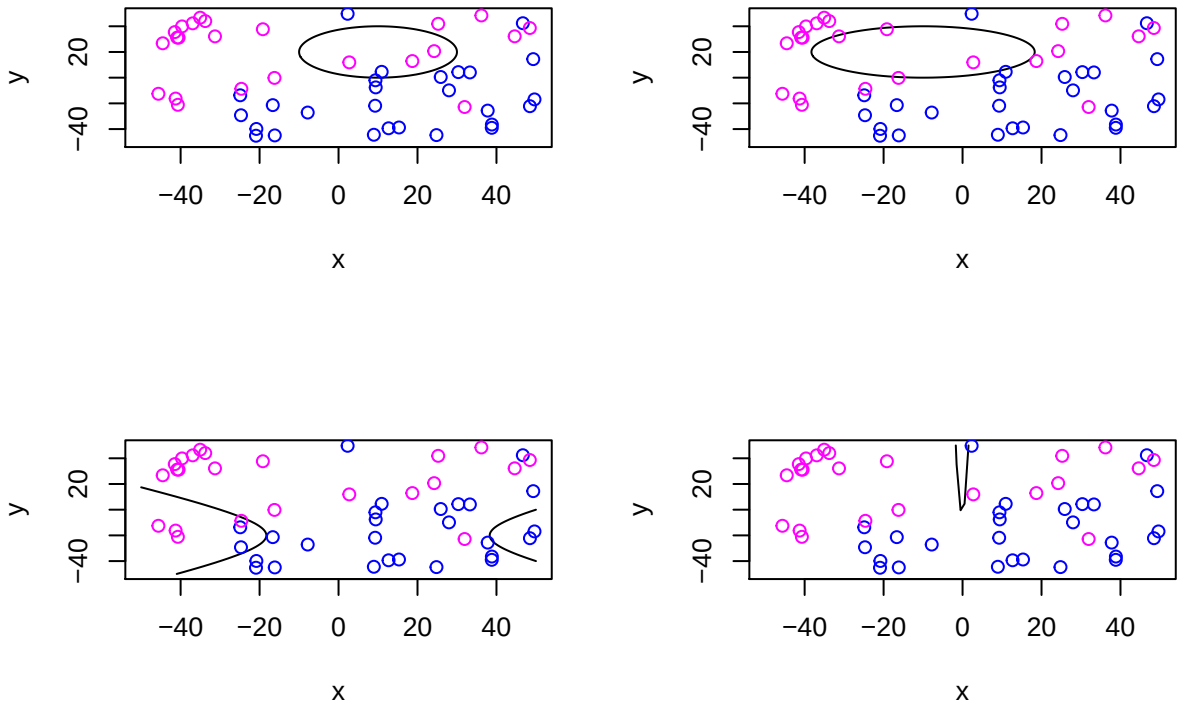
pintar_frontera(f1,datos01,etiquetas_ruido)

pintar_frontera(f2,datos01,etiquetas_ruido)

pintar_frontera(f3,datos01,etiquetas_ruido)

pintar_frontera(f4,datos01,etiquetas_ruido)

Y aquí tenemos las gráficas generadas con estas fronteras y los datos del apartado anterior:



Como podemos observar, las fronteras no consiguen separar los datos etiquetados ya que hay datos de ambas etiquetas a ambos lados de la frontera. Esto se debe a que hemos etiquetado los datos respecto a una recta, no respecto a ninguna de estas fronteras.

Ejercicio 2

Apartado 1

Esta es la función que hemos creado para **ajusta_PLA**. En ella, recorremos nuestra matriz de datos hasta que el signo de uno de las datos no se corresponde con la etiqueta que debería tener, es decir, está mal clasificado. Entonces se reajusta el vector w en función de este dato tal y como se hace en el PLA y se empieza de nuevo a revisar los datos. Estas revisiones se realizan en un máximo de "max_iter". Si llegamos a este máximo sin haber encontrado la solución devolvemos -1 como número de iteraciones. Si por el contrario, en una de las revisiones/iteraciones todos los puntos están bien clasificados, termina la función devolviendo la w y las iteraciones necesarias para haberla encontrado:

```
ajusta_PLA <-function(datos,label,max_iter,vini){
  iteraciones = 0
  for(i in 1:max_iter){
    iteraciones=0
    for(j in 1:nrow(datos)){ ##Utilizamos un for para tener la posibilidad de hacer break()
      if(sign(datos[j,1]*vini[1]+datos[j,2]*vini[2]+datos[j,3]*vini[3]) != sign(label[j])){
```

```
        vini[1] = vini[1] + label[j]*datos[j,1]
        vini[2] = vini[2] + label[j]*datos[j,2]
        vini[3] = vini[3] + label[j]*datos[j,3]
        iteraciones=-1
        break()
      }
    }
    if(iteraciones==0){
      iteraciones=i
      break()
    }
  }
  resultado = c(vini,iteraciones)
}
```

Apartado 2A

Hemos creado la función **calcularAB** donde a partir del vector w nos devuelve las componentes "a" y "b" de la recta:

```
calcularAB<-function(w1,w2,w3){
  a=-w1/w2
  b=-w3/w2
  resultado=c(a,b)
}
```

La llamada a las funciones sería:

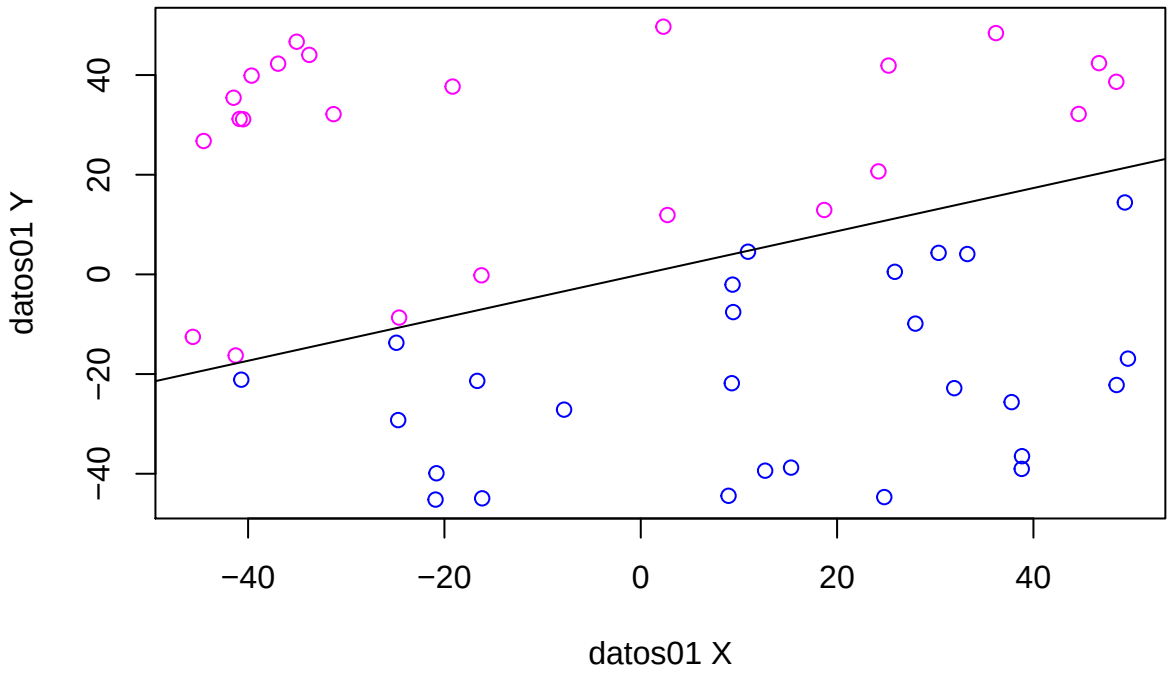
w_ini=c(0,0,0)

pla=ajusta_PLA(datos01_pla,etiquetas,10000,w_ini)

recta_pla=calcularAB(w[1],w[2],w[3])

Y finalmente obtendriamos la gráfica con la nube de puntos junto con la recta que ha calculado el PLA. Además vemos que se han necesitado tan solo 24 iteraciones para que se realice el ajuste:

```
## [1] 24
```



Apartado 2B

Ahora vamos a repetir el apartado anterior pero como w inicial tomaremos 10 w iniciales con valores aleatorios entre 0 y 1. Hemos creado la función **pla_random** que se encarga de llamar a **ajusta_PLA** 10 veces con diferentes w_iniciales como acabamos de comentar. Esta función nos devolvera la media de iteraciones necesarias para encontrar la solución del apartado anterior en los casos que ha tenido éxito junto con las w_iniciales que ha utilizado para ello. Además, nos devuelve las veces que no ha conseguido encontrar la solución y nos pintará una de ellas.

```
pla_random<-function(datos01_pla,etiquetas){
  contador_no_encontrado=0
  contador_encontrado=0
  iteraciones=0
  w_correcta=0
  w_no_encontrada=0
  w_iniciales=c()
  for(i in 1:10){
    w_ini=runif(3,0,1)
    pla=ajusta_PLA(datos01_pla,etiquetas,10000,w_ini)
    if(pla[4]==-1){
      contador_no_encontrado=contador_no_encontrado+1
      w_no_encontrada=c(pla[1],pla[2],pla[3])
    }else{
      iteraciones=iteraciones+pla[4]
      contador_encontrado=contador_encontrado+1
      w_correcta=c(pla[1],pla[2],pla[3])
      w_iniciales=c(w_iniciales,w_ini)
    }
  }
  iteraciones=iteraciones/contador_encontrado
  if(contador_no_encontrado>0){
```

```
    plot(datos01_pla,col=etiquetas+5,xlab="datos01_pla X",ylab="datos01_pla Y")
    recta_pla_r=calcularAB(w_no_encontrada[1],w_no_encontrada[2],w_no_encontrada[3])
    abline(recta_pla_r[2],recta_pla_r[1])
  }
  resultado=c(iteraciones,contador_no_encontrado,w_iniciales)
  resultado
}
```

La llamada a la función se hace con:

resultado=pla_random(datos01_pla,etiquetas)

Y estos son los resultados:

```
## [1] 22.3
```

```
## [1] 0
```

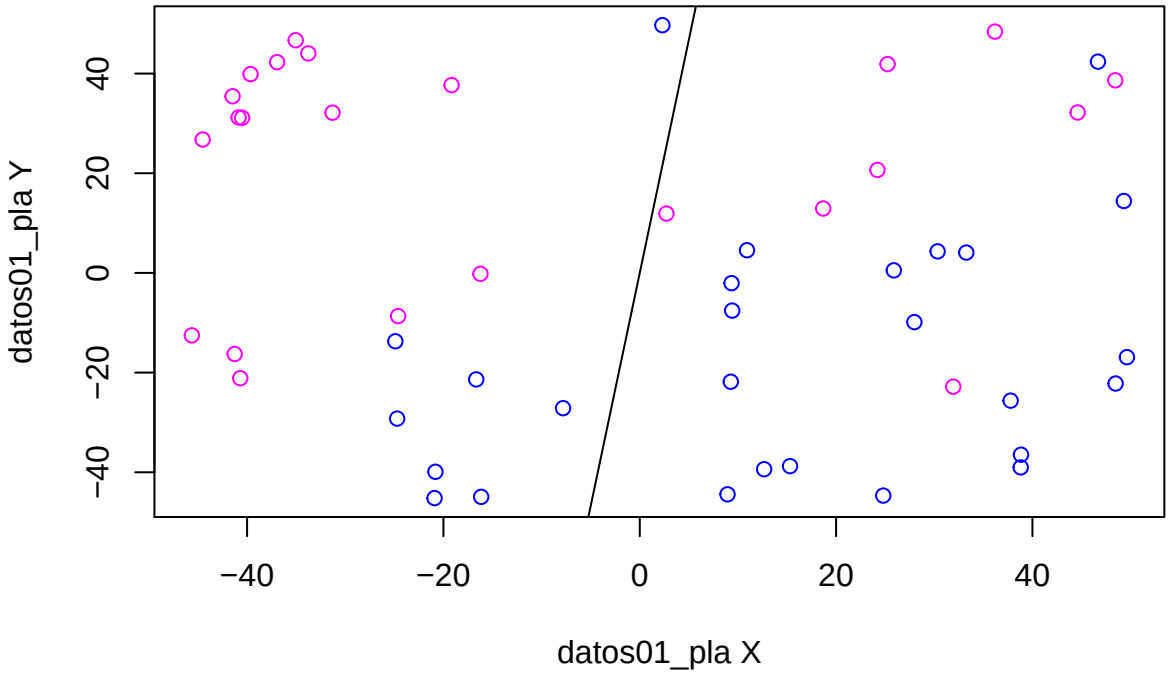
```
##            [,1]       [,2]      [,3]
##  [1,] 0.4532835 0.65588582 0.1777170
##  [2,] 0.8555523 0.56731388 0.6141647
##  [3,] 0.8338344 0.04567028 0.6701204
##  [4,] 0.3165452 0.41614921 0.6310600
##  [5,] 0.7227196 0.23792520 0.3719033
##  [6,] 0.8970900 0.84145801 0.6270702
##  [7,] 0.1133394 0.39963177 0.3918595
##  [8,] 0.5914758 0.18809912 0.8042183
##  [9,] 0.3813533 0.15827643 0.9004539
## [10,] 0.3411453 0.06874332 0.9944410
```

Vemos que la media de las iteraciones ha sido 22.3, muy parecida a cuando utilizamos como w inicial (0,0,0). Además para todas las w iniciales que ha generado, ha encontrado una solución. También vemos las w_iniciales de donde concluimos que tanto para valores cercanos a 0 como cercanos a 1, se encuentra solución

Apartado 3

Ahora vamos a repetir estos dos últimos apartados pero con los datos del apartado 2b del ejercicio 1, es decir, incluyendo ruido en las etiquetas y veremos cómo se comporta nuestro PLA.

Apartado 3A

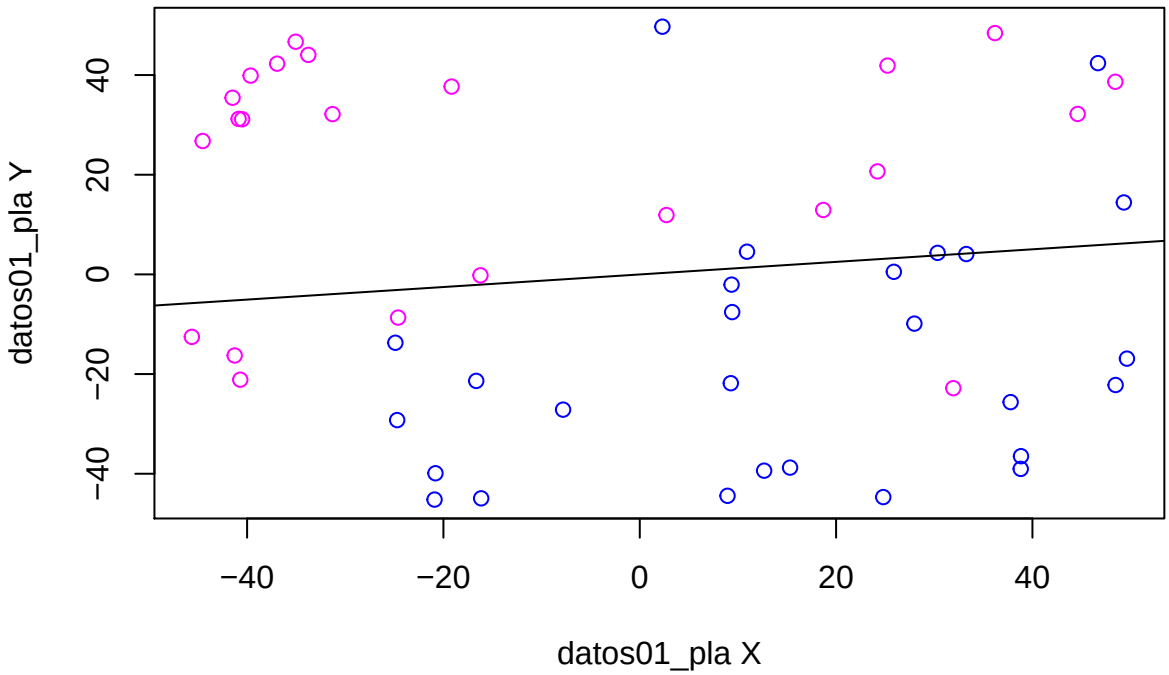



```
## [1] -1
```

Vemos que ahora al utilizar como vector inicial (0,0,0) y haber utilizado las etiquetas con ruido, nuestro PLA no consigue encontrar la solución. Se nos devuleve -1 como número de iteraciones y la recta que pinta no es la solución.

Apartado 3B

Vamos a probar ahora con las 10 w iniciales aleatorias:




```
## Warning in matrix(resultado[3:x], ncol = 3): la longitud de los datos [2]
## no es un submúltiplo o múltiplo del número de columnas [3] en la matriz
```

```
## [1] 10
```

Al igual que en el anterior apartado vemos que para las 10 w iniciales no ha encontrado la solución.

Esto se debe a que al haber creado el etiquetado respecto a una recta y al haberle entonces introducido ruido, ahora no podemos separar los datos etiquetados con una recta, sea cual sea la w inicial que usemos.

Ejercicio 3

Apartado 1

Utilizamos el código proporcionado en Decsai para leer zip.train y zip.test, y obtener los array "grises" y "grises_test" con los números 1 y 5. Hemos creado dos funciones **obtenerImagenes** y **obtenerImagenes_test** que se encargan de transformar los arrays grises y grises_test en listas de las imagenes de 1 y 5 rotadas.

```
obtenerImagenes<-function(){
  imagenes = list()
  for(i in 1:ndigitos_train){
    imagenes[[i]] = grises[i,,16:1]
  }
  imagenes
}

obtenerImagenes_test<-function(){
  imagenes = list()
  for(i in 1:ndigitos_test){
    imagenes[[i]] = grises_test[i,,16:1]
  }
```

```
  imagenes
}
```

Así la llamada a las funciones sería:

imagenes_train=obtenerImagenes()

imagenes_test=obtenerImagenes_test()

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : número de items leídos no es múltiplo del número de columnas
```

Apartado 2

A continuación vamos a extraer la intensidad promedio y la simetría de cada una de estas imagenes que hemos generado. Para ello utilizamos la función **fsimetria** que nos da la simetría de una imagen, **calcularMedia** que nos da la intensidad promedio de la imagen sumando todos sus valores y haciendo la media, **obtenerMatrizSimetriaMedia** que recorre las imagenes que hemos creado y le calcula a cada una su simetría y media, devolviendo una matriz con todos estos valores, y finalmente una función **cambiar_digitos_etiquetas** que transforma el vector "digitos" de 1 y 5 en etiquetas de 1 y -1 (cambiando 5 por -1) para poder tratarlo como etiquetas positivas y negativas.

```
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  -sum(A)
}

calcularMedia<-function(Matriz){
  media=sum(Matriz)/256
}

calculaMediaySimetria<-function(A){
  n1=calcularMedia(A)
  n2=fsimetria(A)
  resultado = c(n1,n2)
}

obtenerMatrizSimetriaMedia<-function(imagenes){
  matriz = c()
  n=length(imagenes)
  for(i in 1:n){
    r = calculaMediaySimetria(imagenes[[i]])
    matriz = rbind(matriz,r)
  }
  matriz
}

cambiar_digitos_etiquetas<-function(digs,tam){
  for(i in 1:tam){
    if(digs[i]==5)
      digs[i]=-1
  }
  digs
}
```

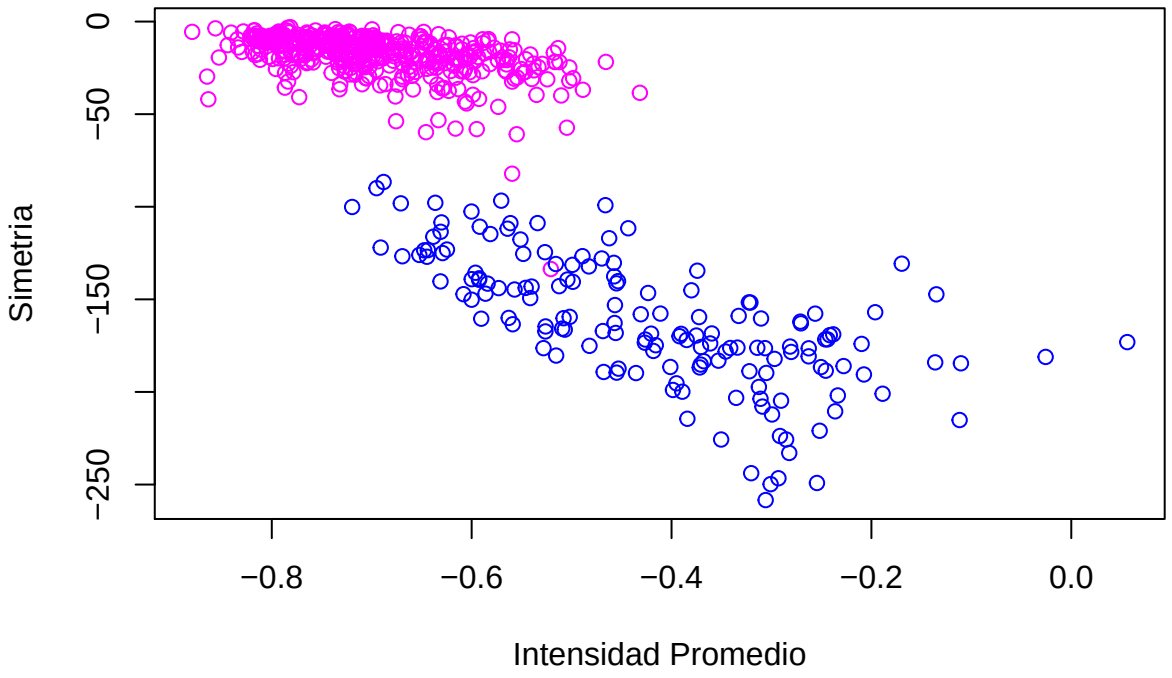
Las llamadas a las funciones serían:

m_train=obtenerMatrizSimetriaMedia(imagenes_train)

m_test=obtenerMatrizSimetriaMedia(imagenes_test)

etiquetas_train=cambiar_digitos_etiquetas(digitos_train,599)

Utilizando estas funciones y leyendo el archivo Zip.train dibujamos la gráfica X=Intensidad Promedio y Y=Simetría para estos datos:



Apartado 3

La transformación SVD para ajustar la regresión lineal la hemos implementado mediante las funciones **Regress_Lin** y **pseudo_inversa**. **Regress_Lin** nos devuelve la w para construir nuestra recta a partir de los datos de intensidad y promedio calculados, y las etiquetas asignadas. Para ello multiplica la pseudo_inversa de la matriz de datos (calculada con **pseudo_inversa**) por las etiquetas:

```
Regress_Lin<-function(datos,label){
  cadena=replicate(nrow(datos),1)
  datos=cbind(datos,cadena)
  pm=pseudo_inversa(datos)
  cad1=pm[1,]*label
  cad2=pm[2,]*label
  cad3=pm[3,]*label
  w1=sum(cad1)
  w2=sum(cad2)
  w3=sum(cad3)
  w=c(w1,w2,w3)
}

pseudo_inversa<-function(X){
  aux=t(X)%*%X
  X_svd=svd(aux)
```

```
  pseudo=X_svd$u%*%diag(1/X_svd$d)%*%t(X_svd$v)
  pseudo_X=pseudo%*%t(X)
  pseudo_X
}
```

Además, hemos creado la función **calcularE** que nos calcula el error en función del porcentaje de puntos mal clasificados de nuestro modelo respecto de las etiquetas:

```
calcularE<-function(w,datos,etiquetas,tam){
  cadena=replicate(tam,1)
  datos=cbind(datos,cadena)
  media=0
  for(i in 1:tam){
  if(sign(datos[i,1]*w[1]+datos[i,2]*w[2]+datos[i,3]*w[3])!=sign(etiquetas[i]))
    media=media+1
}
  media=media*100/tam
  media
}
```

Así, las llamadas a las funciones serían:

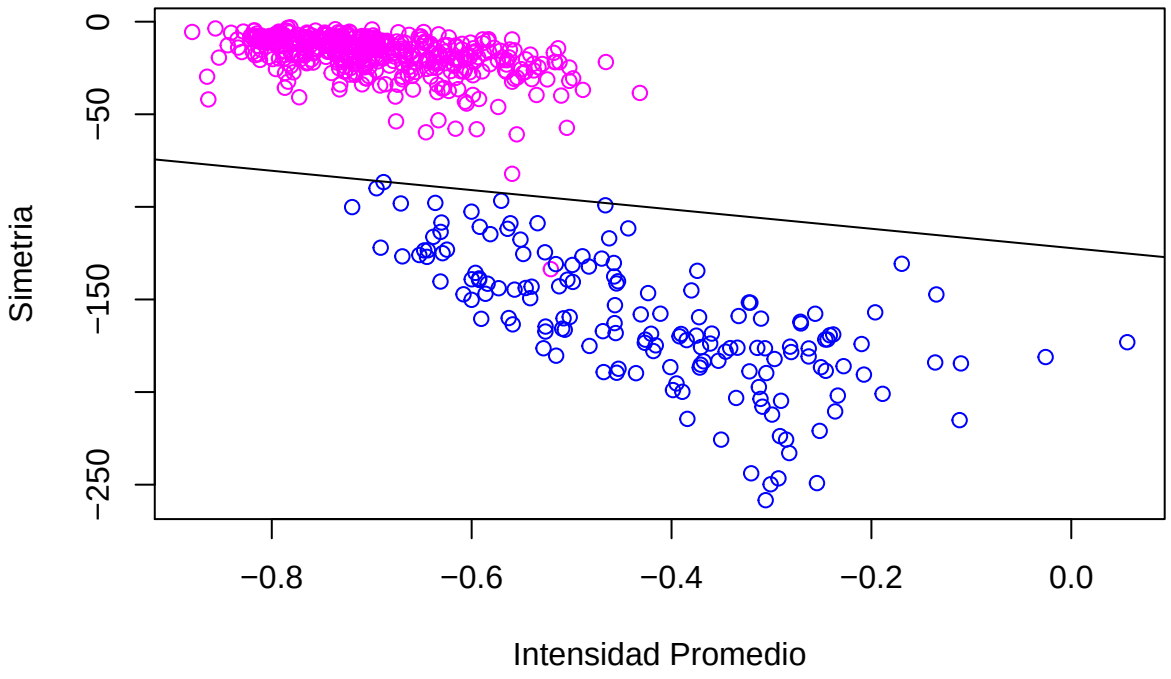
w=Regress_Lin(m_train,etiquetas_train)

recta_train=calcularAB(w[1],w[2],w[3])

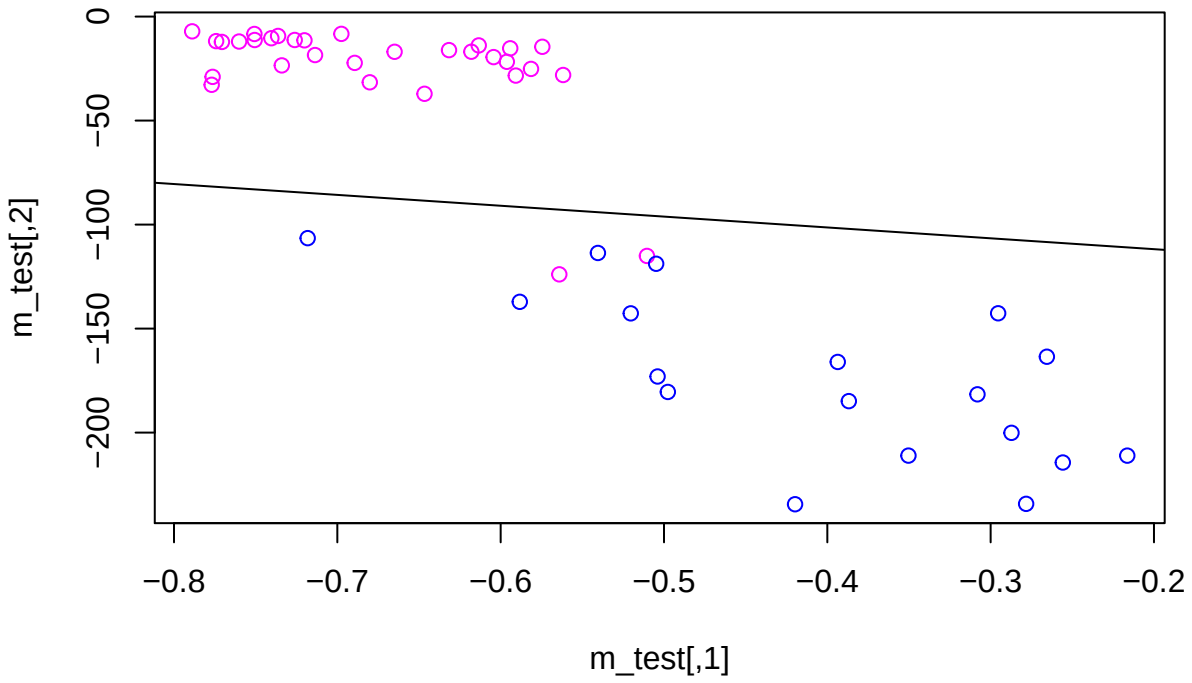
error_train=calcularE(w,m_train,etiquetas_train,599)

error_test=calcularE(w,m_test,etiquetas_test,49)

Esta es la gráfica con los datos y la recta que acabamos de calcular:



Y aqui vemos como separa esta recta para el conjunto de test:



Y aqui tenemos los errores tanto para el conjunto de entrenamiento como para el de test:

```
## [1] 0.1669449
```

```
## [1] 4.081633
```

Vemos que como era de esperar el error fuera de la muestra (4%) es mayor que dentro de la muestra. No obstante, la regresión lineal presenta un buen resultado tanto dentro como fuera de la muestra.

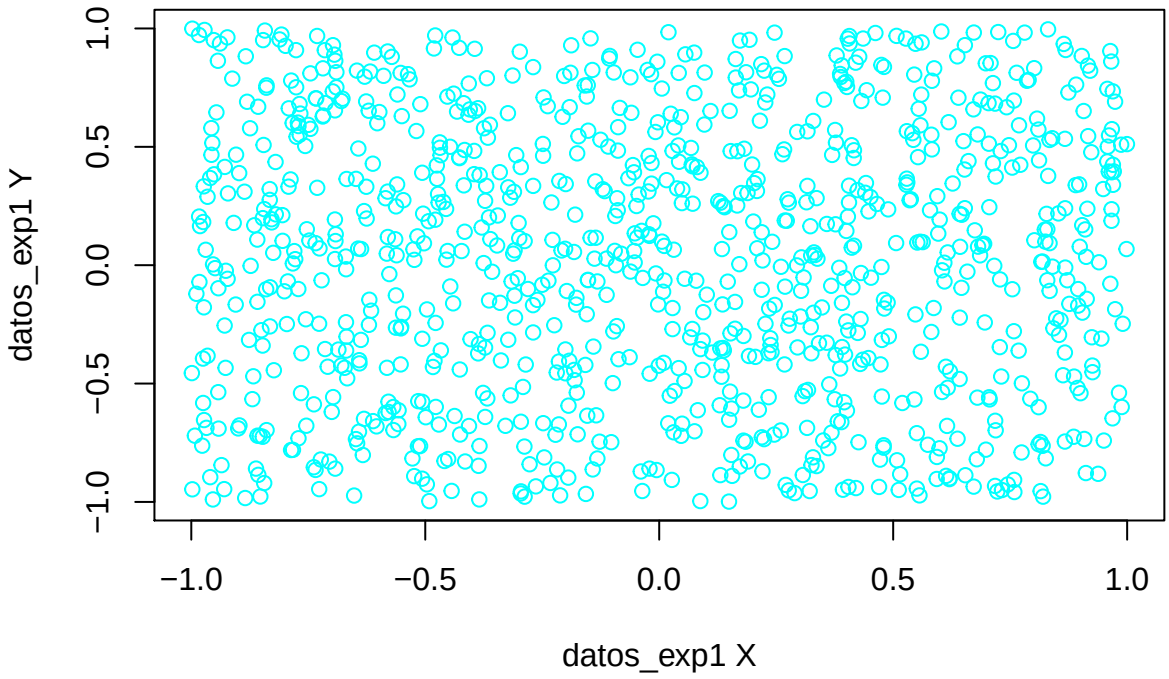
Apartado 4

Experimento 1

Apartado A

Generamos la muestra de entrenamiento con

datos_exp1=simula_unif(1000,2,c(-1,1))



Apartado B

Utilizamos la siguiente función para generar las etiquetas:

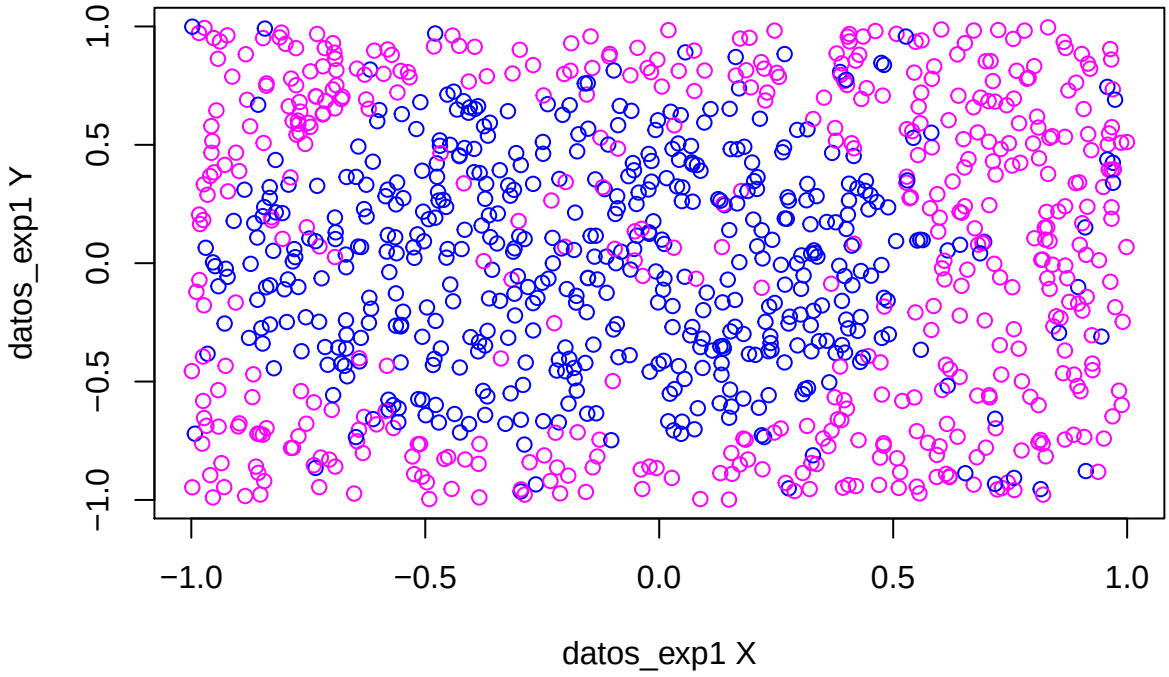
```
generarEtiquetaExp1<-function(datos){
    sign((datos[1]+0.2)^2 + datos[2]^2 - 0.6)
}
```

Y las llamadas a la funciones para generar las etiquetas con ruido serían:

etiquetas_exp1=apply(datos_exp1,1,generarEtiquetaExp1)

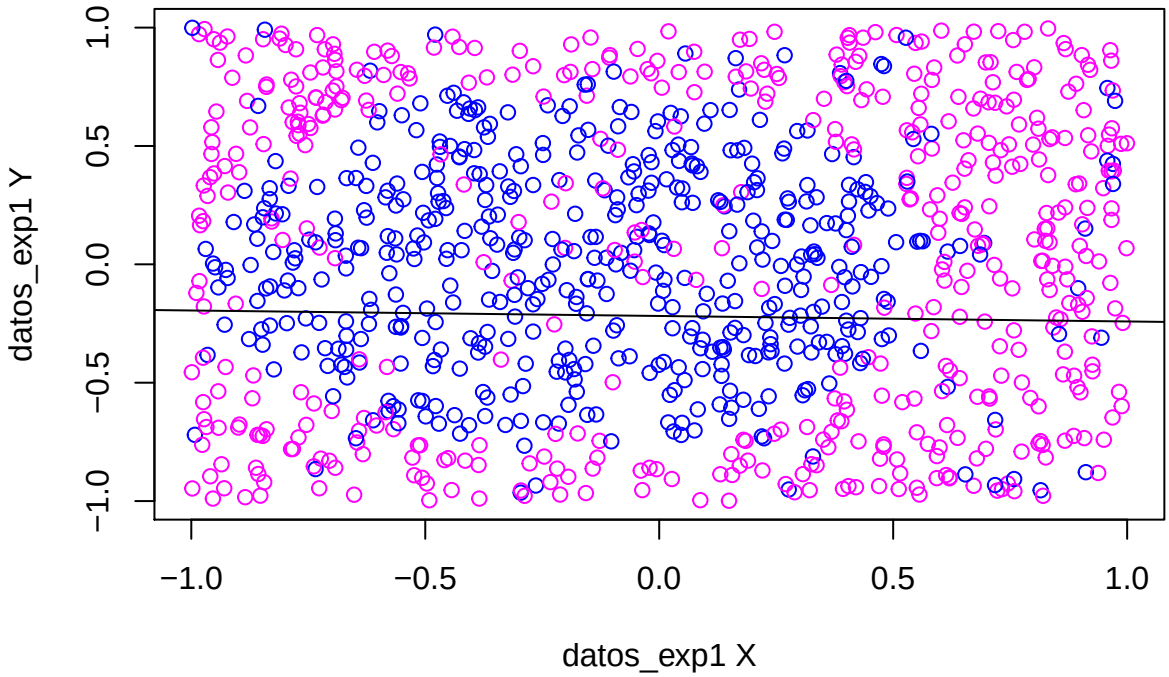
etiquetas_exp1=modificarEtiquetas(etiquetas_exp1,10)

Aquí tenemos los datos etiquetados:



Apartado C

Ahora vamos a intentar un ajuste con nuestro modelo de regresión lineal. Se ha hecho una pequeña modificación en las funciones **Regress_Lin** y **calcularE** para que ahora no modifiquen la matriz de datos. Esta es la gráfica del ajuste:



Y este es el error del ajuste:

```
## [1] 41.5
```

Vemos que el error es muy elevado, más del 40%

Apartado D

Ahora vamos a repetir estos apartados 1000 veces, generando 1000 muestras diferentes y calcularemos el error medio tanto dentro como fuera de la muestra:

```
## numeric(0)
```

```
## numeric(0)
```

Apartado E

Vemos que el error dentro y fuera de la muestra es muy similar y a la vez muy elevado, casi un 40%. Esto se debe basicamente a que no hemos generado las etiquetas respecto a una recta y ahora no nos es posible clasificar los datos con una recta.

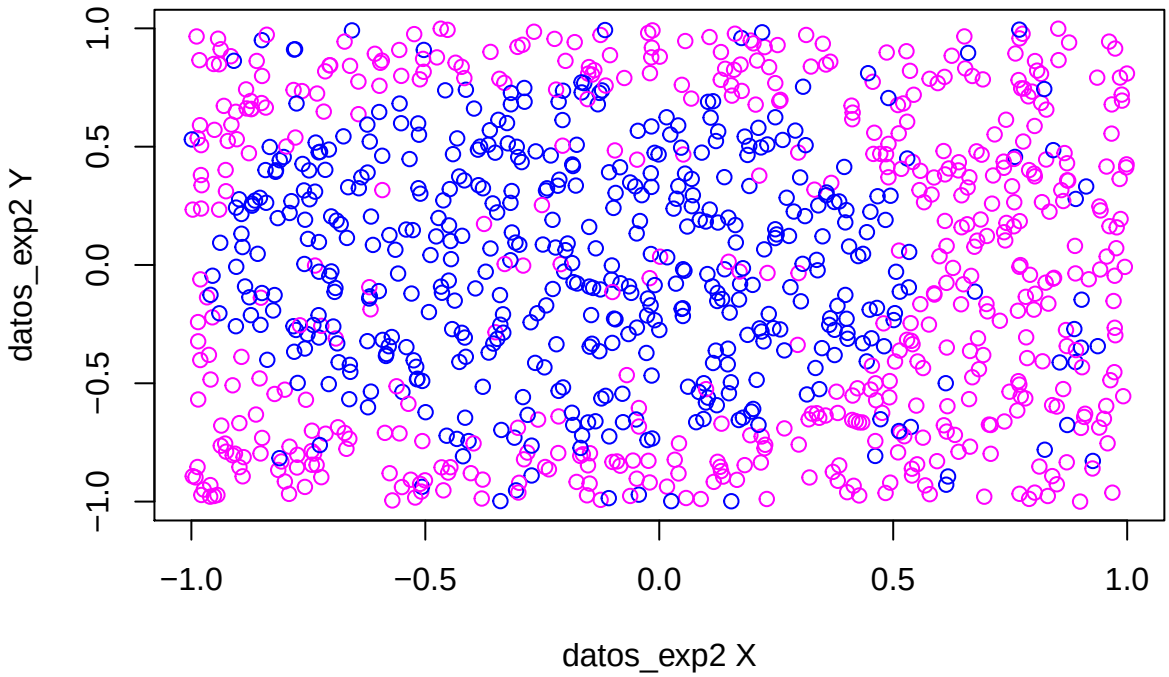
Experimento 2

Apartado A

Para cambiar el vector de caracteristicas hemos creado la función **modificarVector** que modifica la matriz de x,y a 1,x,y,xy,x^2 ,y^2:

```
modificarVector<-function(datos){
  cadena=replicate(1000,1)
  datos=cbind(cadena,datos)
  cadena=datos[,2]*datos[,3]
  datos=cbind(datos,cadena)
  cadena=datos[,2]*datos[,2]
  datos=cbind(datos,cadena)
  cadena=datos[,3]*datos[,3]
  datos=cbind(datos,cadena)
  datos
}
```

Además hemos modificado las funciones de **Regress_Lin** y **calcularE** para que ahora tengan en cuenta las 6 columnas de la matriz de datos. Realizando de nuevo el ajuste mediante el modelo de regresión lineal obtenemos para la siguiente gráfica:



El siguiente error en el ajuste, dentro de la muestra:

```
## [1] 14.4
```

Podemos ver que el error ha decrecido mucho respecto del error que obteniamos cuando estabamos intentando un ajuste con una recta.

Apartado B

Al igual que hicimos en el anterior experimento vamos a repetirlo 1000 veces y vamos a calcular el error medio tanto dentro como fuera de la muestra:

```
## numeric(0)
```

```
## numeric(0)
```

Apartado C

En ambos el error medio es de algo más del 14%. Si bien puede parecer un error elevado, tenemos que tener en cuenta que se han utilizado etiquetas con ruido y que se ha logrado una gran mejora respecto al experimento 1 ya que ahora se ajustan mejor los datos al etiquetado que hemos creado.

Bonus

Apartado A

Usamos funciones que hemos visto anteriormente tales como **Regress_Lin** o **calcularE** pero esta vez para los datos:

datos_bonus = simula_unif(100,2, c(-10,10))

recta_bonus = simula_recta(c(-10,10))

etiquetas_bonus = generarEtiquetasMatriz(datos_bonus,recta_bonus[1],recta_bonus[2])

Lo realizamos 1000 veces y calculamos la media del error. Este es el resultado:

```
## [1] 4.009
```

Podemos ver que tenemos un error medio del 4%. Es un buen error considerando que las rectas que determinaban las etiquetados se han creado de forma aleatoria.

Apartado B

Realizamos lo mismo pero esta vez utilizando 1000 puntos:

datos_bonus = simula_unif(1000,2, c(-10,10))

Y este es el resultado:

```
## [1] 3.723
```

Vemos que obtenemos un error medio del 3.7%. Es algo inferior que el de dentro de la muestra, incluso teniendo ahora 1000 puntos.

Apartado C

De forma similar a los apartados anteriores con 10 puntos obtenemos el vector de pesos con regresión lineal y lo utilizamos como vector inicial en **ajusta_PLA**. Se realiza 1000 veces y calculamos la media del número de iteraciones necesarias para que converja el PLA. Se ha hecho una modificación sobre **ajusta_PLA** para que en cada iteración empiece desde un punto aleatorio. Este es el resultado:

```
## [1] 6.938
```

Podemos ver que de media se han necesitado unas 7 iteraciones para que converja el PLA. Este buen resultado se debe al buen ajuste inicial que se consigue con la regresión inicial, llegando en ocasiones a que el ajuste inicial era solución al problema.