Práctica 1 Aprendizaje Automático

Jacinto Carrasco Castillo

Generación y visualización de datos

Antes de realizar cualquier operación que involucre elementos aleatorios estableceremos la semilla para que los datos y ejecuciones en las que se basan las conclusiones del informe sean reproducibles.

```
set.seed(314159)
```

Para hacer paradas en cada paso, se define la función `stp()`

```
stp <- function(){
  cat("Pulse una tecla para continuar")
  t<- scan()
}
```

Construir una función `lista = simula_unif (N, dim, rango)` que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios uniformes en el intervalo rango..

```
simula_unif <- function(N, dim, rango){
  # Tomamos N*dim muestras de una uniforme de rango dado
  lista <- matrix(runif(N*dim, min =  rango[1], max = rango[2]), N, dim)
  return(lista)
}
```

Construir una función `lista = simula_gaus(N, dim, sigma)` que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios gaussianos de media 0 y varianzas dadas por el vector $sigma$

```
simula_gauss <- function(N, dim, sigma){
  #Tomamos N*dim elementos de una normal, tomando la desviación estándar del vector sigma
  lista <- matrix(rnorm(N*dim, sd = sigma), N, dim, byrow=TRUE)
  return(lista)
}
```

En este caso estamos rellenando la matriz por filas. Haciéndolo de esta manera, y puesto que sigma es un vector cuya dimensión debería ser 1 o coincidir con la dimensión de los vectores (dim), el número generado se genera con la desviación típica correspondiente a la de su dimensión.

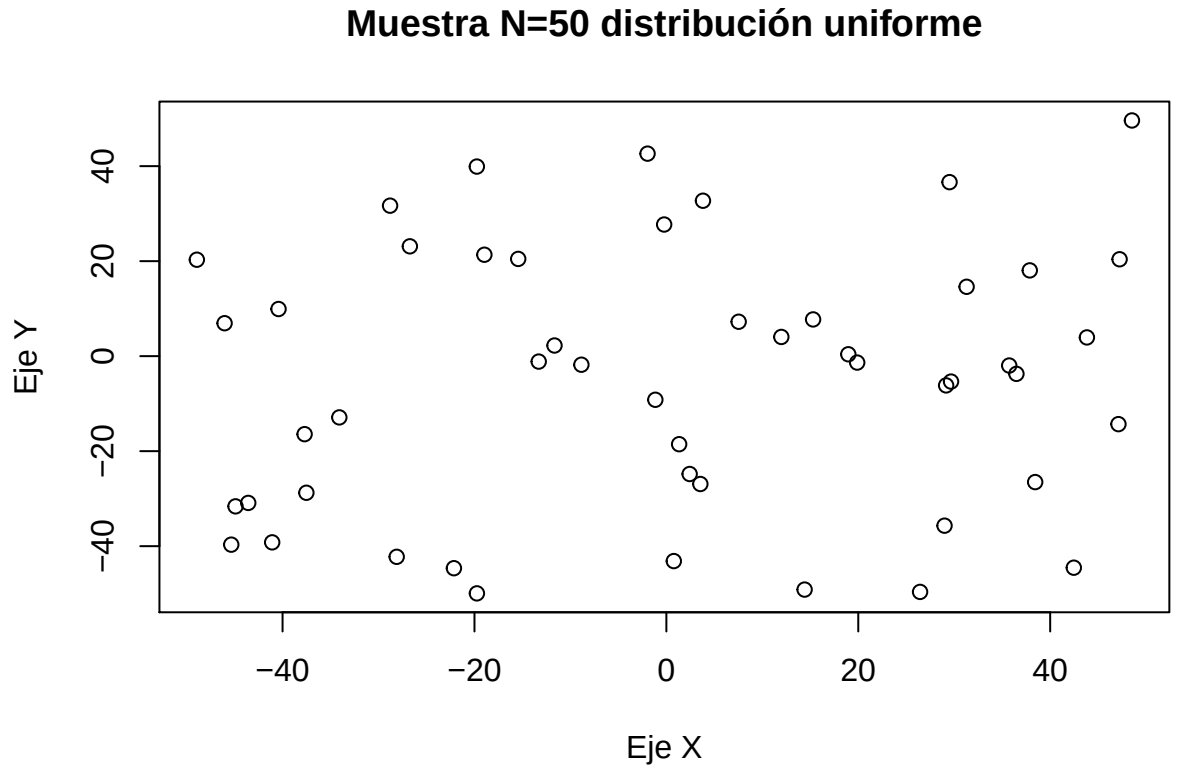
Suponer $N = 50, dim = 2, rango = [-50, +50]$ en cada dimensión para generar una lista de puntos de una distribución uniforme. Generar puntos también de una distribución normal con $N = 50, dim = 2$ y $sigma = [5, 7]$. Dibujar una gráfica de la salida de ambas funciones.

```
lista_unif <- simula_unif(50, 2, c(-50, 50))
lista_gauss <- simula_gauss(50, 2, c(5,7))
stp()
```

```
## Pulse una tecla para continuar
```

Para representar los valores, ponemos en el eje X la primera columna del conjunto de datos y en el eje Y la segunda columna. Si comparamos la representación de la distribución normal con la distribución uniforme,

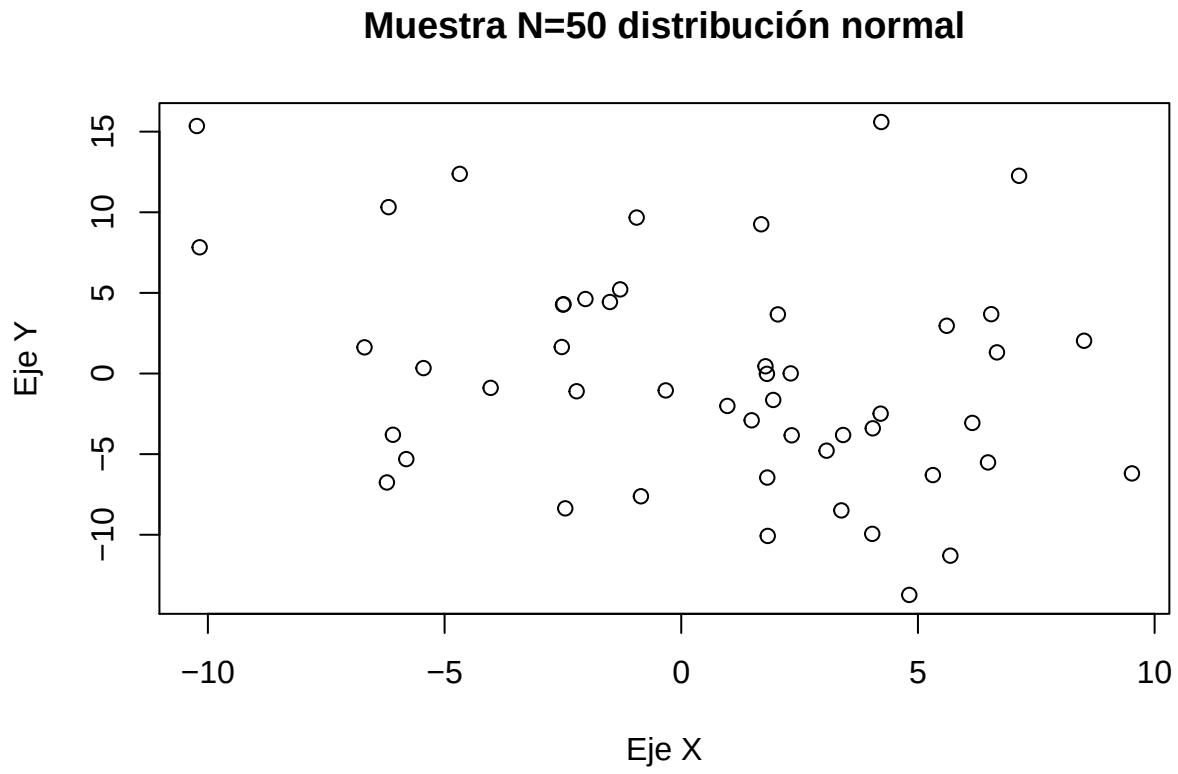
```
plot(lista_unif[,1], lista_unif[,2], xlab="Eje X", ylab="Eje Y",
     main="Muestra N=50 distribución uniforme" )
```




```
stp()
```

```
## Pulse una tecla para continuar
```

```
plot(lista_gauss[,1],lista_gauss[,2], xlab="Eje X", ylab="Eje Y",
     main="Muestra N=50 distribución normal")
```

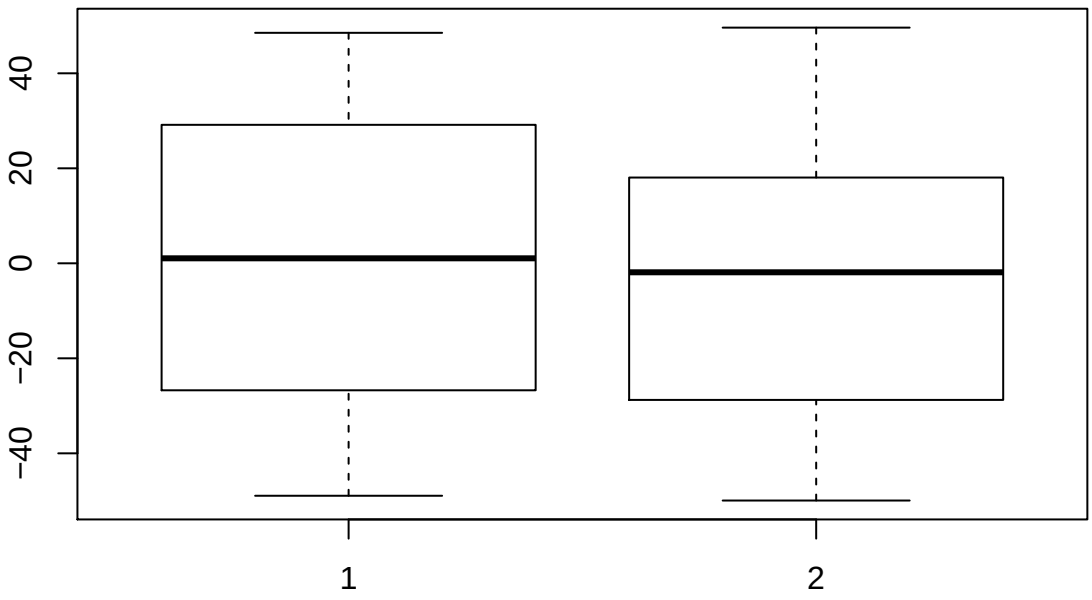



```
stp()
```

```
## Pulse una tecla para continuar
```

Si mostramos ahora los diagramas de cajas con bigote para cada una de las dimensiones tanto en la distribución uniforme como en la distribución de Gauss, vemos que en la distribución uniforme ambos diagramas son muy parecidos (como era de esperar pues la distribución es la misma), mientras que en la distribución normal, el rango intercuartílico de la segunda dimensión es más amplio, ya que tiene desviación típica

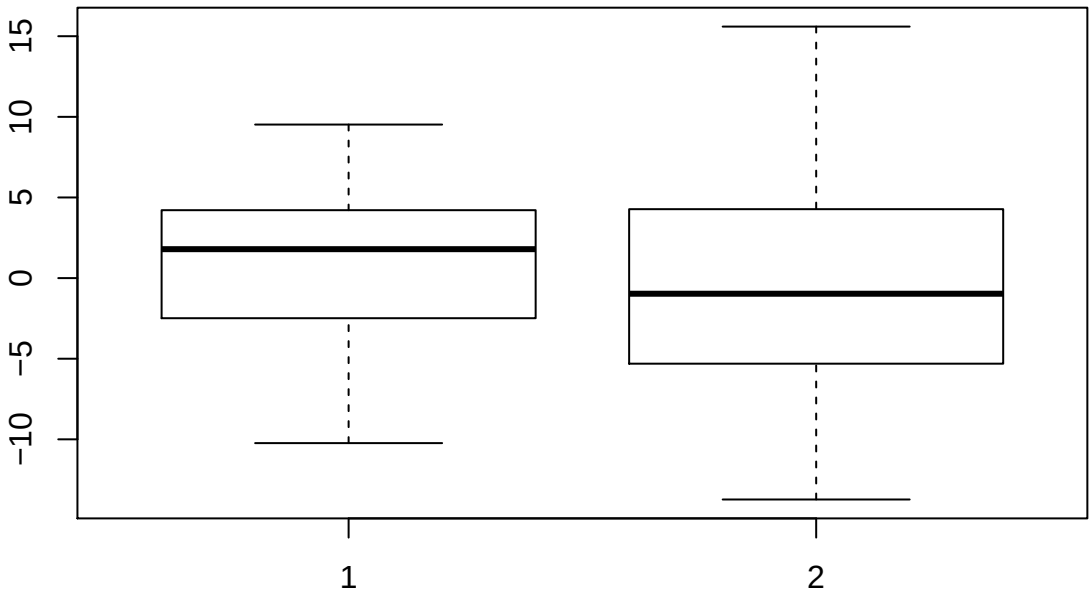
```
boxplot(lista_unif[,1],lista_unif[,2])
```



```
stp()
```

```
## Pulse una tecla para continuar
```

```
boxplot(lista_gauss[,1],lista_gauss[,2])
```




```
stp()
```

```
## Pulse una tecla para continuar
```

Construir la función `v=simula_recta(intervalo)` que calcula los parámetros, $v = (a, b)$ de una recta aleatoria, $y = ax + b$, que corte al cuadrado [-50, 50] $\times$ [-50, 50] (Ayuda: Para calcular la recta simular las coordenadas de dos puntos dentro del cuadrado y calcular la recta que pasa por ellos)

Generamos otra muestra de tamaño 2 de una distribución uniforme y calculamos la recta que pasa por estos dos puntos.

```
simula_recta <- function(intervalo){
  puntos <- simula_unif(2,2,intervalo)
  a <- (puntos[2,2]-puntos[1,2])/(puntos[2,1]-puntos[1,1])
  b <- puntos[1,2] - a * puntos[1,1]
  return(c(a,b))
}
```

Etiquetado de muestra uniforme

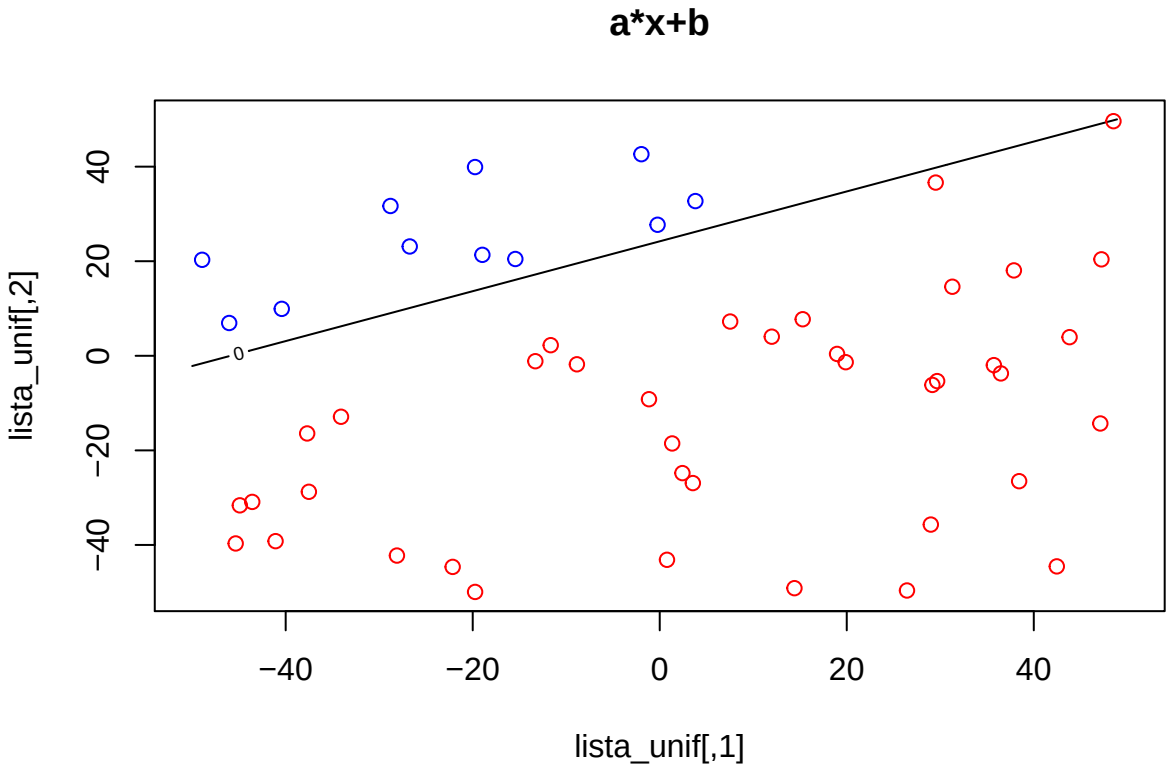
Generar una muestra 2D de puntos usando `simula_unif()` y etiquetar la muestra usando el signo de la función $f(x, y) = y - ax - b$ de cada punto a una recta simulada con `simula_recta()`. Mostrar una gráfica con el resultado de la muestra etiquetada junto con la recta usada para ello.

Generamos los coeficientes de una recta y etiquetamos la muestra de puntos generada anteriormente. Para realizar el etiquetado, applicamos `apply` por filas, generando un vector de etiquetas con el signo de la función.

```
intervalo = c(-50, 50)
coefs <- simula_recta(intervalo)
a <- coefs[1]
b <- coefs[2]

f <- function(X){
  return(X[2] - a*X[1] -b)
}

etiqueta = apply(lista_unif, 1, function(X) sign(f(X)))
```

Escribimos una función que dibuje una función en $\mathbb{R}^2$ en un cierto intervalo, usando la función `contour` para pintar también posteriormente las funciones cuadráticas. El argumento `add` permitirá añadir un gráfico a un `plot` existente.

```
draw_function <- function(interval_x, interval_y, f, levels = 0, col = 1, add = FALSE){
  x <- seq(interval_x[1],interval_x[2],length=1000)
  y <- seq(interval_y[1],interval_y[2],length=1000)
  z <- outer(x,y, f) # Matriz con los resultados de hacer f(x,y)

  # Levels = 0 porque queremos pintar f(x,y)=0
  contour(x,y,z, levels=0, col = col, add = add)
}

draw_function(c(-50,50), c(-50,50),function(x,y) return(y - a*x -b))
title(main="a*x+b", ylab = "lista_unif[,2]", xlab = "lista_unif[,1]")
points(lista_unif[,1], lista_unif[,2], col = (etiqueta+3))
```

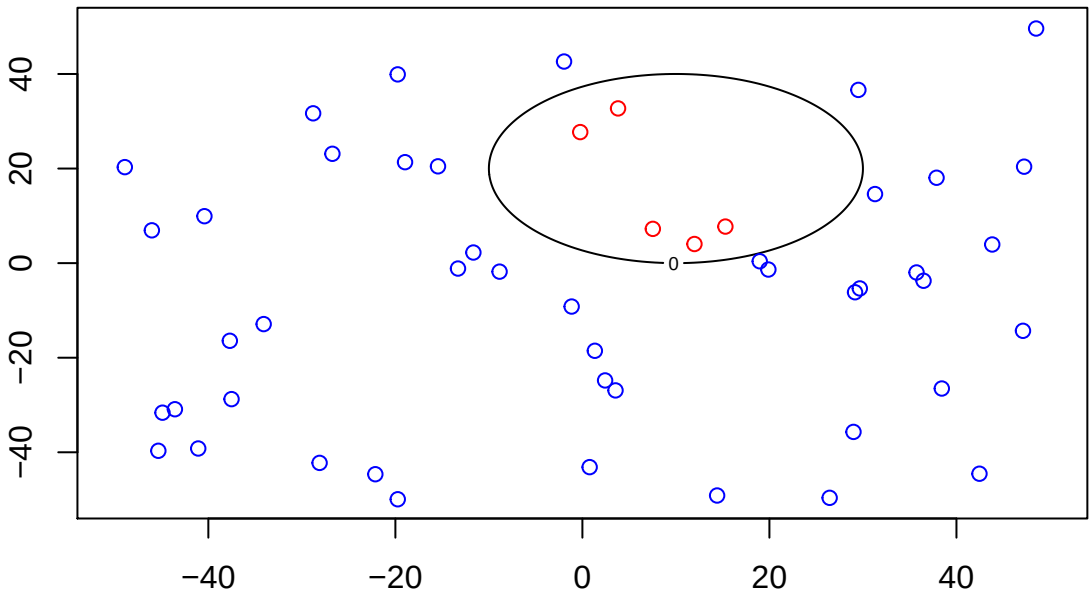


Como vemos en la gráfica, los puntos rojos (etiquetados con -1) se sitúan debajo de la gráfica ylos azules por encima.

Usar la muestra generada en el apartado anterior y etiquetarla con +1,-1 usando el signo de cada una de las siguientes funciones

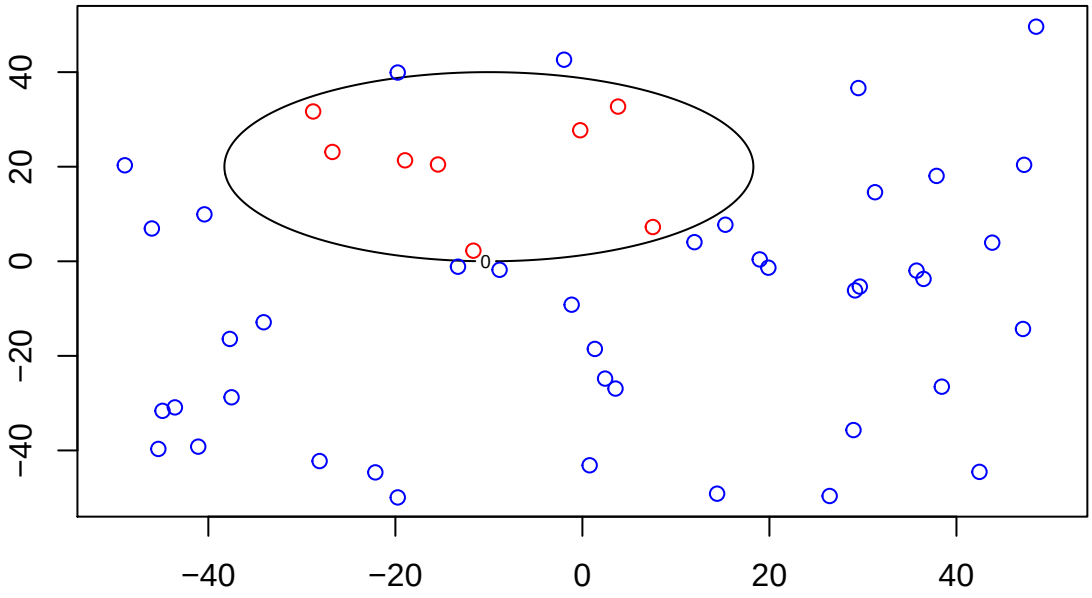
Aquí es donde verdaderamente entra en juego la función `contour`, pues nos permite representar una función cuadrática de manera sencilla. Para realizar el etiquetado, lo hacemos como para el caso de la recta, tomando el signo de la evaluación de la función en cada dato de la muestra.
En este caso la mayoría de datos cae fuera de la elipse corresponidente a la función.

```
f_1 <- function(x,y){
  return((x - 10)^2 + (y - 20)^2 - 400)
}

etiqueta_1 = apply(lista_unif, 1, function(X) sign(f_1(X[1],X[2])))

draw_function(c(-50,50), c(-50,50), f_1)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_1+3))
```



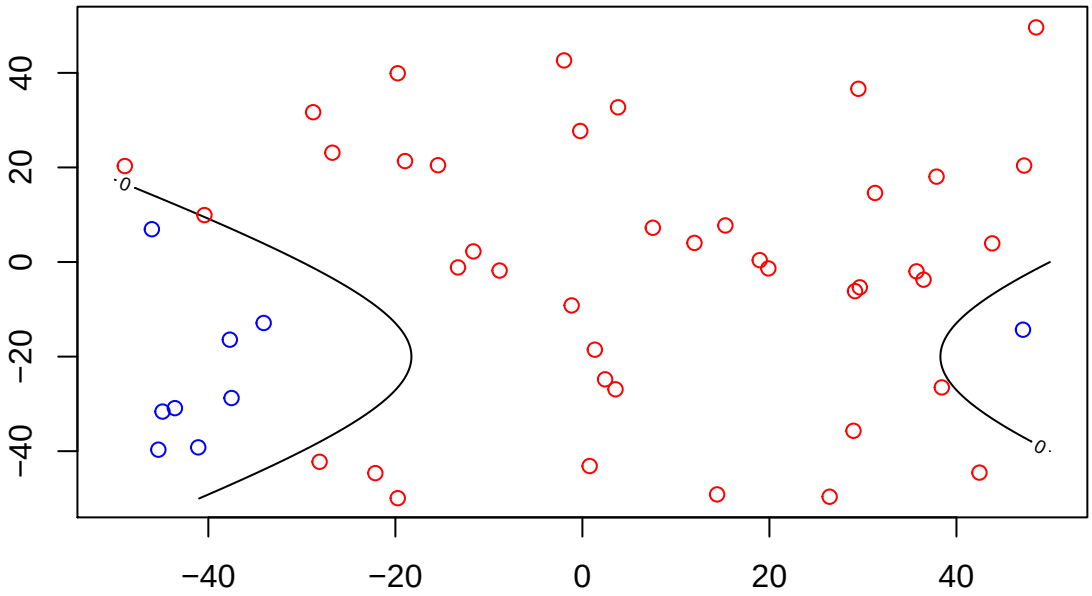
Para esta segunda función, también la mayoría de datos cae fuera de la cuádrica y es una elipse, como podíamos deducir de sus coeficientes.

```
f_2 <- function(x,y){
  return(0.5*(x + 10)^2 + (y - 20)^2 - 400)
}

etiqueta_2 = apply(lista_unif, 1, function(X) sign(f_2(X[1],X[2])))

draw_function(c(-50,50), c(-50,50),f_2)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_2+3))
```



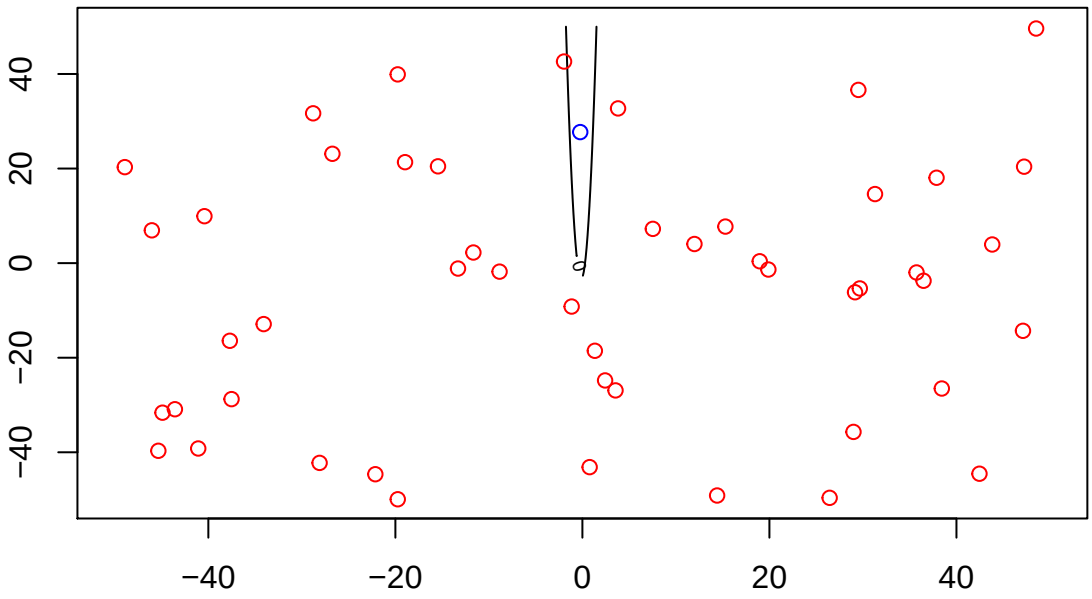
En esta hipérbola, la mayoría de los datos caen fuera, o bien en la parte izquierda.

```
f_3 <- function(x,y){
  return(0.5*(x - 10)^2 - (y + 20)^2 - 400)
}

etiqueta_3 = apply(lista_unif, 1, function(X) sign(f_3(X[1],X[2])))

draw_function(c(-50,50), c(-50,50), f_3)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_3+3))
```



En esta parábola sólo uno de los datos cae por encima de la gráfica. Esto dará sin dudas problemas a la hora de clasificar, pues no tenemos información suficiente sobre los datos con etiqueta 1.

```
f_4 <- function(x,y){
  return(y - 20*x^2- 5*x + 3)
}

etiqueta_4 = apply(lista_unif, 1, function(X) sign(f_4(X[1],X[2])))

draw_function(c(-50,50), c(-50,50), f_4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_4+3))
```



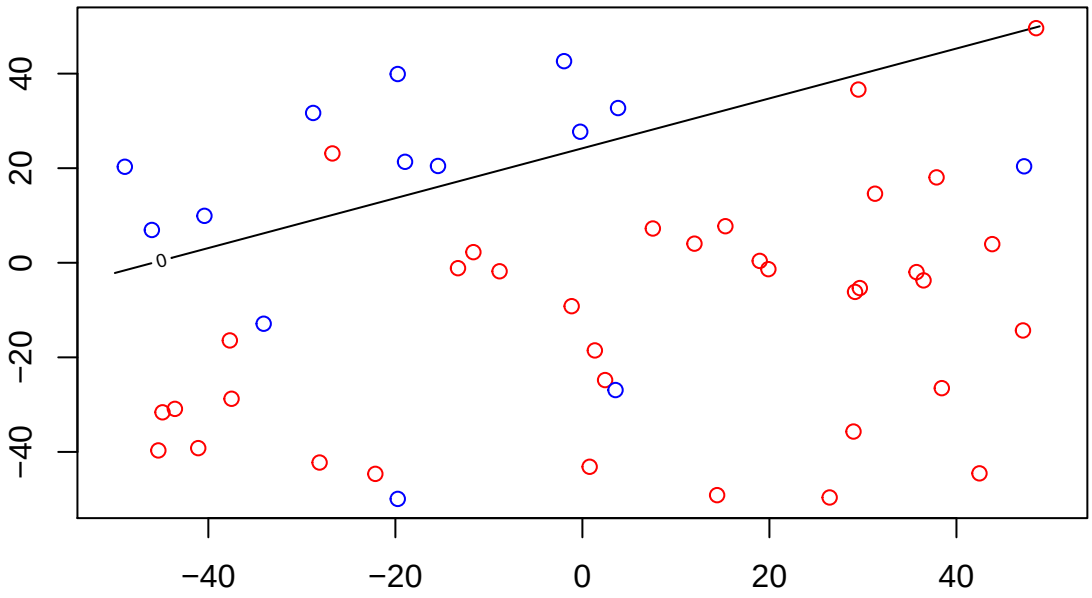
De la representación gráfica de estas funciones y los datos clasificados según ellas, se extrae que no son separables linealmente, por lo que el algoritmo del Perceptron no convergerá, y será necesario una nueva aproximación para resolverlo.

Considerar de nuevo la muestra etiquetada en el apartado 6. Modifique las etiquetas de un 10 % aleatorio de muestras positivas y otro 10% aleatorio de negativas.

```
modify_rnd_bool_subvector <- function(v, perc = 0.1){
  #Almacenamos una copia para poder saber qué puntos debemos cambiar en el original.
  mod_v <- v

  #Contamos el número de puntos que corresponde cambiar para hacerlo sólo si hay alguno
  length_change = round(length(which(v == 1))*perc)
  if( length_change >= 1){
    to_change <- sample(which(v == 1), length_change )
    mod_v[to_change] = -1
  }

  length_change = round(length(which(v == -1))*perc)
  if( length_change >= 1){
    to_change <- sample(which(v == -1), length_change )
    mod_v[to_change] = 1
  }

  return(mod_v)
}
```

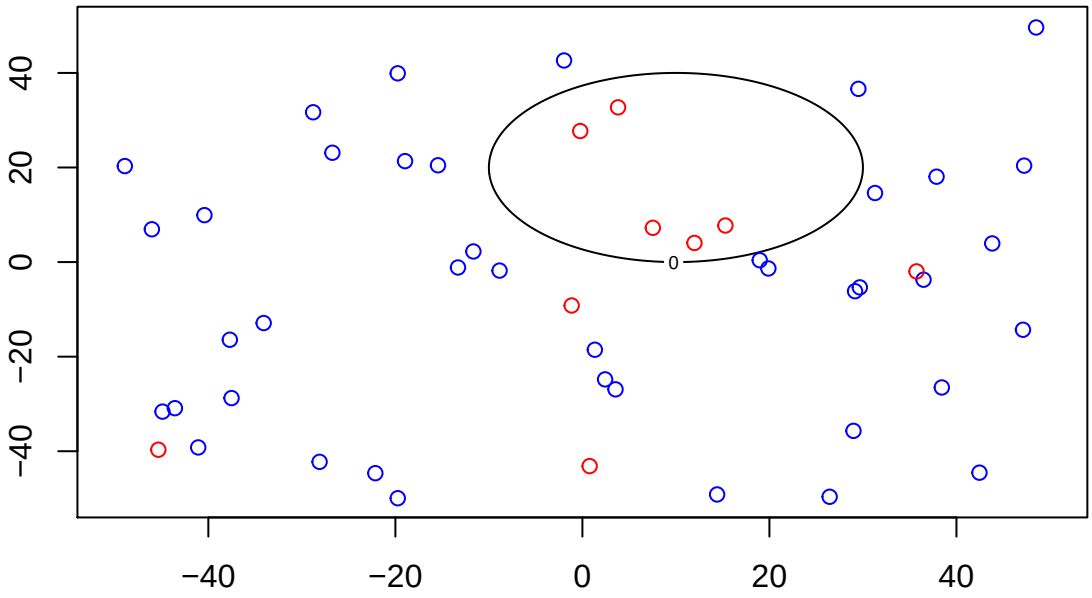
Visualice los puntos con las nuevas etiquetas y la recta del apartado 6

```
etiqueta_mod <- modify_rnd_bool_subvector(etiqueta)

draw_function(c(-50,50), c(-50,50), function(x,y) y-a*x-b)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```

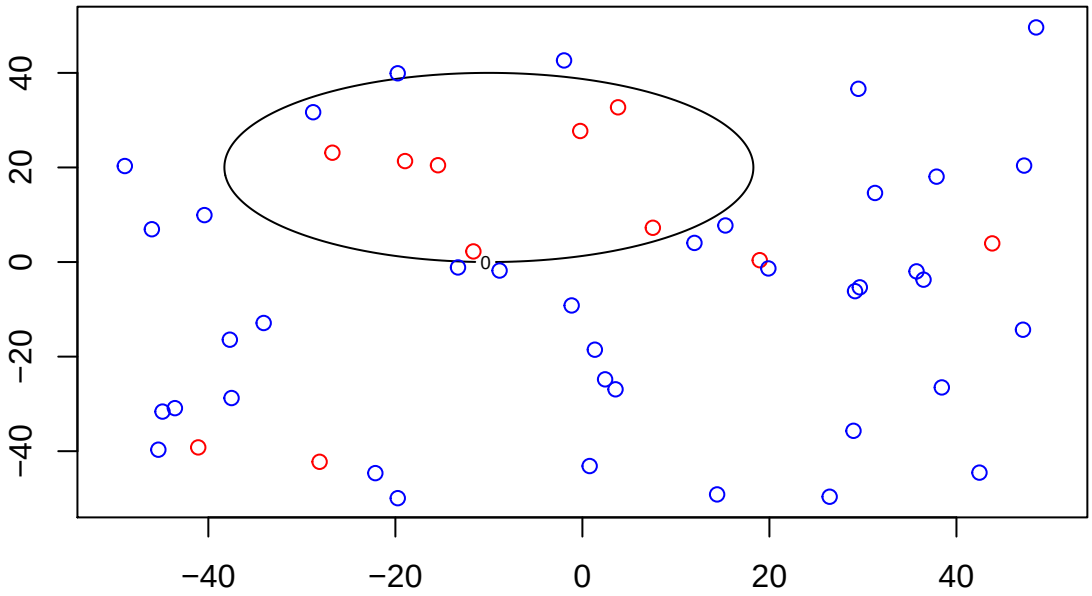


En una gráfica aparte visualice nuevo los mismos puntos pero junto con las funciones del apartado 7

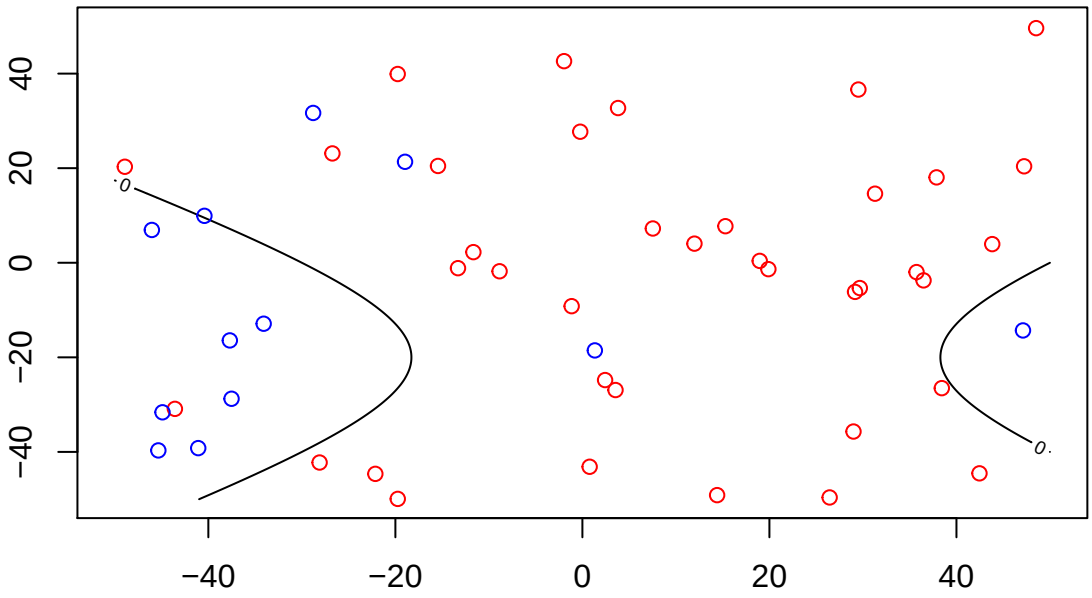
```
etiqueta_mod_1 <- modify_rnd_bool_subvector(etiqueta_1)

draw_function(c(-50,50), c(-50,50), f_1)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```



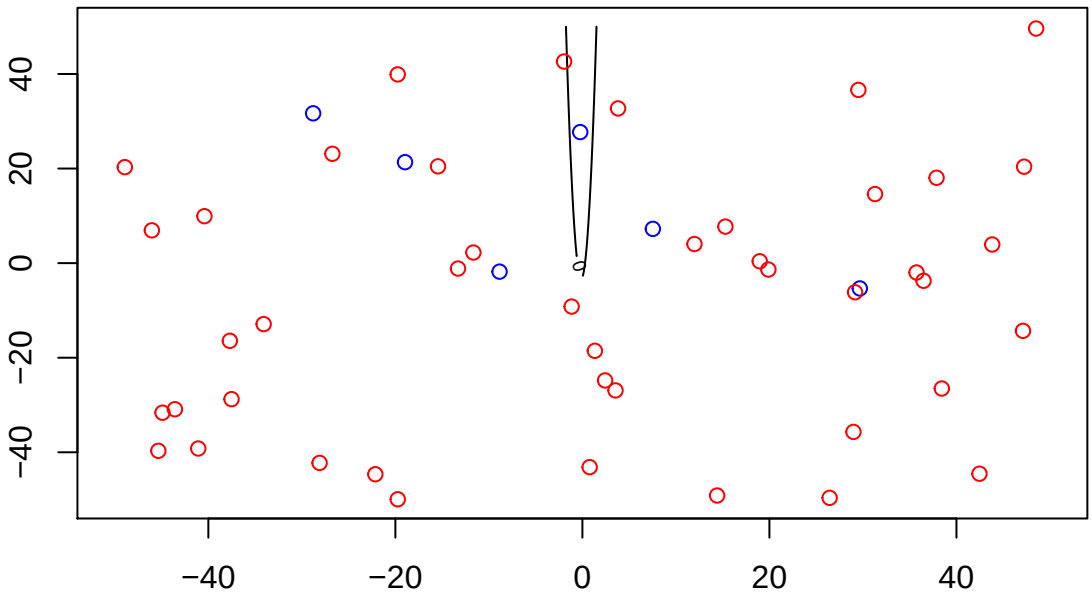
```
etiqueta_mod_2 <- modify_rnd_bool_subvector(etiqueta_2)

draw_function(c(-50,50), c(-50,50), f_2)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_2+3))
```




```
etiqueta_mod_3 <- modify_rnd_bool_subvector(etiqueta_3)

draw_function(c(-50,50), c(-50,50), f_3)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_3+3))
```




```
etiqueta_mod_4 <- modify_rnd_bool_subvector(etiqueta_4)

draw_function(c(-50,50), c(-50,50), f_4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_4+3))
```



En el proceso de cambio vemos que si juntamos que la clase donde hay menos puntos, hay pocos, y que cambiamos el 10% de valores, la clasificación se antoja aún más difícil, pues ya por ejemplo en el tercer y cuarto caso apenas se distinguen áreas claramente definidas.

Ajuste del algoritmo Perceptron

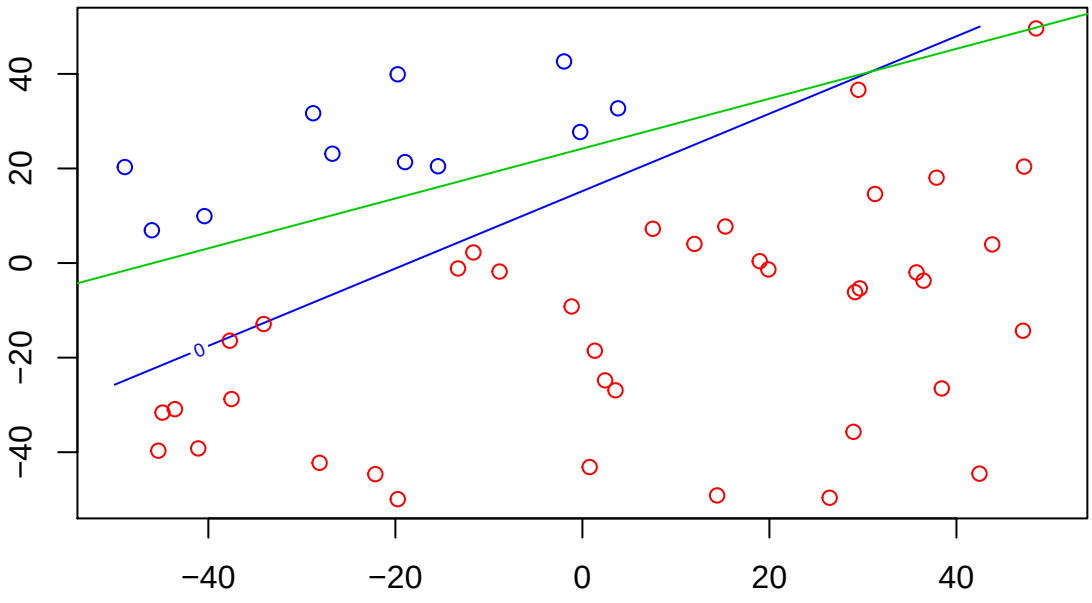
Implementar la función `sol = ajusta_PLA(datos, label, max_iter, vini)` que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada datos es una matriz donde cada item con su etiqueta está representado por una fila de la matriz, label el vector de etiquetas (cada etiqueta es un valor +1 o-1), max_iter es el número máximo de iteraciones permitidas y vini el valor inicial del vector. La salida sol devuelve los coeficientes del hiperplano.

```
ajusta_PLA <- function(datos, label, max_iter, vini, draw_iterations = FALSE){
  sol <- vini
  iter <- 0
  changed <- TRUE

  while ( iter < max_iter && changed){
    # Comprobaremos en cada vuelta si hemos hecho algún cambio
    changed <- FALSE
    for( inner_iter in 1:nrow(datos) ){
      # Añadimos coeficiente independiente
      x <- c(datos[inner_iter,],1)

      if( sign(crossprod(x,sol)) != label[inner_iter]){
        sol <- sol + label[inner_iter] * x
        changed <- TRUE
      }
    }
```

```
    #Guardamos cada una de las iteraciones en un archivo para mostrarla posteriormente
    if(draw_iterations){
      draw_function(c(-50,50), c(-50,50), function(x,y) y +sol[1]/sol[2]*x
                    +sol[3]/sol[2], col = 4)
      points(lista_unif[,1], lista_unif[,2], col = (label+3))
      Sys.sleep(0.1)
    }
    iter <- iter+1
  }

  #Normalizamos la solución.
  sol <- sol/sol[length(sol)-1]

  #Devolvemos el hiperplano obtenido y el número de iteraciones realizadas
  return( list( hiperplane = sol, iterations = iter))
}

sol_pla <- ajusta_PLA(lista_unif,etiqueta, 200,rep(0,3))$hiperplane
```

La parte para dibujar y guardar las iteraciones no está demasiado relacionado con el algoritmo, pero puesto que se pide que se dibujen las iteraciones en determinados puntos, la he incluido. La normalización de la solución por la coordenada Y del vector solución se podría realizar simplemente a la hora de pintar la gráfica, pero así son datos más interpretables.
Con los datos generados en el ejercicio anterior, ejemplo del algoritmo PLA, que da la solución en azul, mientras que se pinta con `abline` la recta con la que realizamos la clasificación. Se aprecia que efectivamente se ha resuelto, como mucho, las 200 iteraciones con las que se ha ejecutado. Pese a diferir de la recta original, no se encuentra ningún dato en las regiones de diferencia, por lo que no teníamos ninguna información para descartar la solución aportada

```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol_pla[1]*x+sol_pla[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta+3))
abline(b, a, col=3)
```



Ejecutar el algoritmo PLA con los valores simulados en apartado.6 del ejercicio.5.2, inicializando el algoritmo con el vector cero y con vectores de números aleatorios en [0, 1] (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado.

Creamos un vector numérico de tamaño 10 que vamos rellenando con las iteraciones de sucesivas ejecuciones del algoritmo PLA para vectores iniciales aleatorios (salvo el primero que es $(0, 0, 0)$). No uso una función vectorial sino un bucle para ello para que el vector aleatorio generado sea distinto en cada iteración.

```
iterations <- numeric()
length(iterations) <- 10

iterations[1] <- ajusta_PLA(lista_unif,etiqueta, 10000,rep(0,3))$iterations

for(i in 2:10){
  v_ini <- runif(3)
  iterations[i] <- ajusta_PLA(lista_unif,etiqueta, 10000, v_ini)$iterations
}

print(c("Media de iteraciones ", mean(iterations)))
```

```
## [1] "Media de iteraciones " "55.3"
```

Como vemos, se llevan a cabo 59 iteraciones de media. Es un resultado aceptable teniendo en cuenta que la recta que separa a los dos conjuntos de datos no pasa por el origen. Por ejecuciones con distintos conjuntos de datos, la convergencia cuando ocurre esto es mucho mayor, ya que una de las componentes del hiperplano (el término independiente) ya sería correcto. En cambio, para conjuntos de datos donde el término independiente tiene un mayor valor absoluto, el número de iteraciones también es mayor.

Ejecutar el algoritmo PLA con los datos generados en el apartado.8 del ejercicio.5.2, usando valores de 10, 100 y 1000 para `max_iter`. Etiquetar los datos de la muestra usando la función solución encontrada y contar el número de errores respecto de las etiquetas originales. Valorar el resultado.

Para contar los errores, generamos un vector con los signos de la aplicación de la función solución a los datos y contamos los que sean distintos de las etiquetas.

```
count_errors <- function(f, datos, label){
  #Conteo de errores
  signes <- apply(datos, 1, function(x) return(sign(f(c(x,1)))))
  v <- signes != label
  return(length(v[v]))
}
```

Al haber hecho la función para contar errores de esta manera (por si en el futuro tuviéramos funciones que no fuesen lineales) necesitamos pasar de un vector (hiperplano) a una función lineal.

```
hiperplane_to_function <- function( vec ){
  f <- function(x){
    return( crossprod(vec,x) )
  }
```

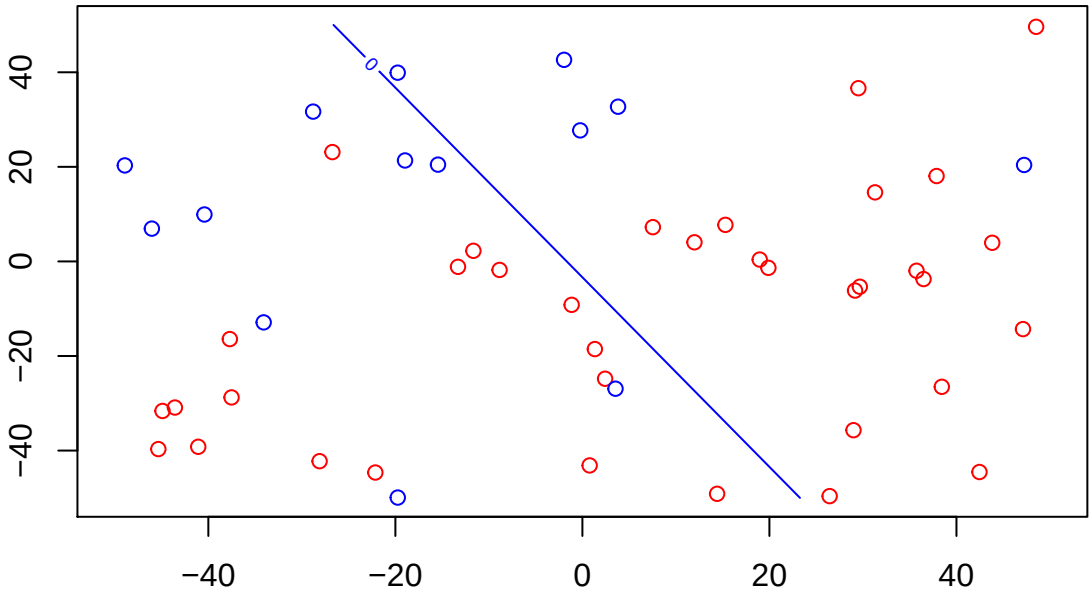
```
  return(f)
}
```

Conteo de errores

```
sol <- ajusta_PLA(lista_unif,etiqueta_mod, 10,rep(0,3))$hiperplane
cat("Errores con 10 iteraciones", count_errors(hiperplane_to_function(sol),
                                               lista_unif, etiqueta_mod ))
```

```
## Errores con 10 iteraciones 28
```

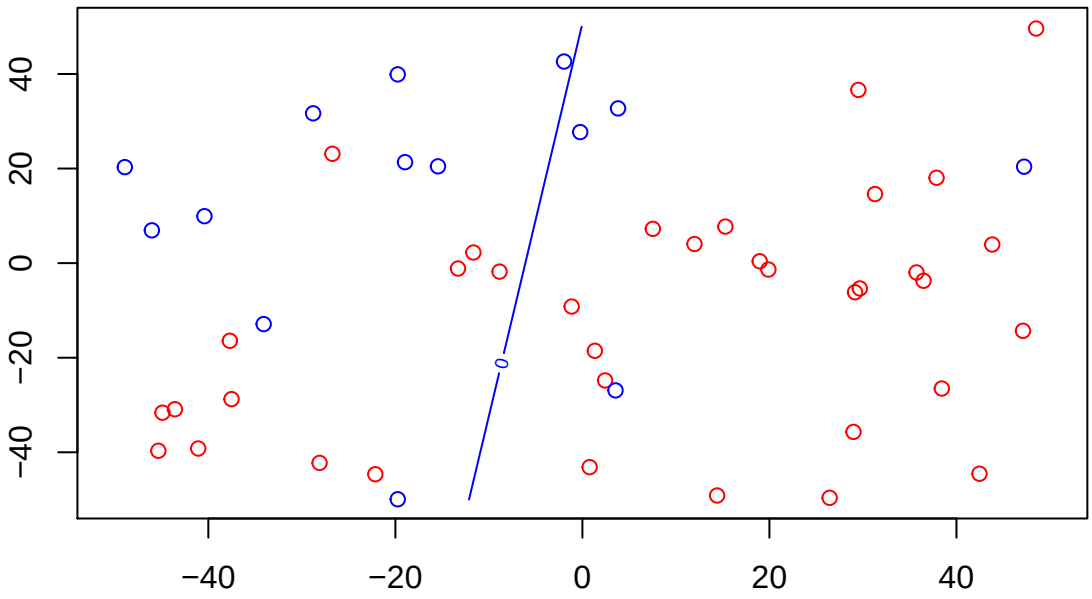
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```




```
sol <- ajusta_PLA(lista_unif,etiqueta_mod, 100,rep(0,3))$hiperplane
cat("Errores con 100 iteraciones",count_errors(hiperplane_to_function(sol),
                                               lista_unif, etiqueta_mod ))
```

```
## Errores con 100 iteraciones 16
```

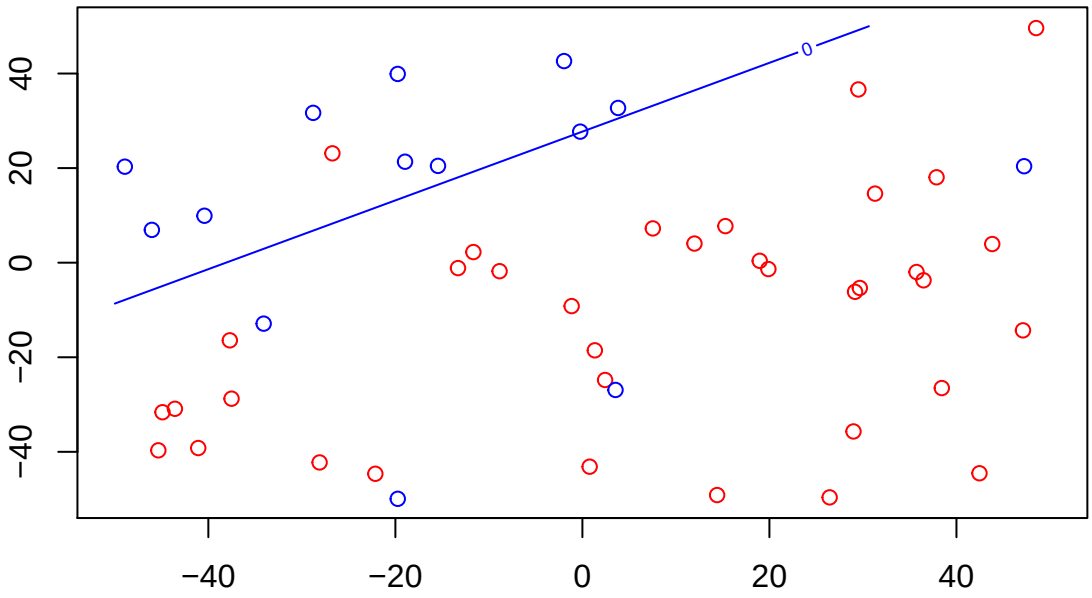
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```




```
sol <- ajusta_PLA(lista_unif,etiqueta_mod, 1000,rep(0,3))$hiperplane
cat("Errores con 1000 iteraciones",count_errors(hiperplane_to_function(sol),
                                                lista_unif, etiqueta_mod ))
```

```
## Errores con 1000 iteraciones 5
```

```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```



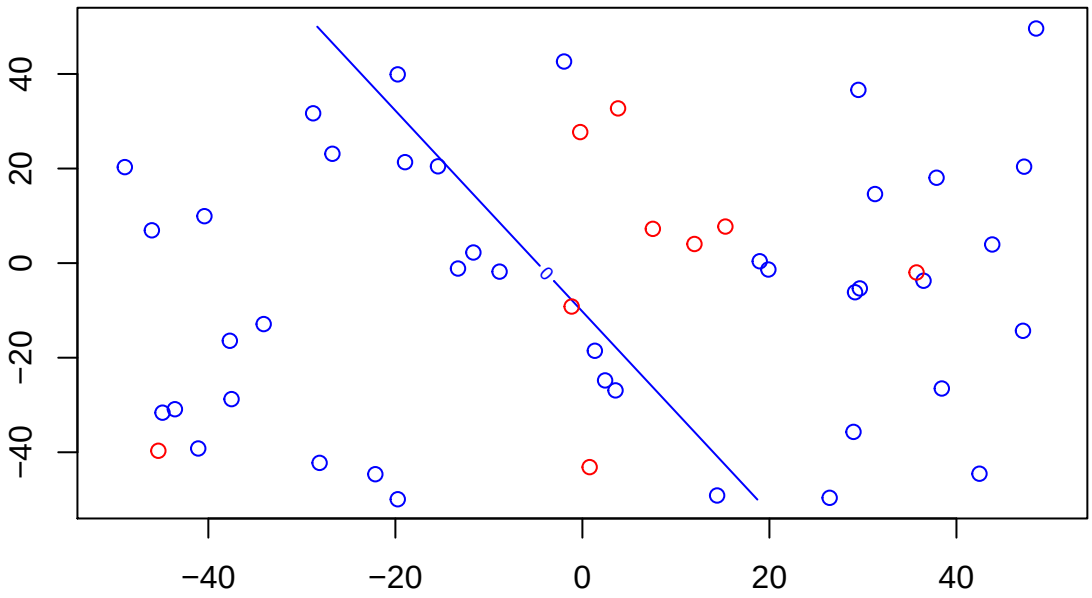
Ahora, con 1000 iteraciones llega a una solución equivalente a la del apartado anterior, aunque lógicamente con errores pues los datos no son separables. Aún así, es la mejor solución que podíamos encontrar ya que sólo están mal clasificados los datos cuya etiqueta hemos cambiado, y así se halla el mínimo de errores posibles. En cambio, realizando menos iteraciones tenemos más errores que estos cinco que hemos cambiado. Esto se explica porque el algoritmo también tratará de clasificar bien los datos que hacen el conjunto no separable, lo que entorpece el proceso.

Repetir el análisis del punto anterior usando la primera función del apartado.7 del ejercicio 5.2.

```
sol <- ajusta_PLA(lista_unif,etiqueta_mod_1, 10,rep(0,3))$hiperplane
cat("Errores con 10 iteraciones", count_errors(hiperplane_to_function(sol),
                                               lista_unif, etiqueta_mod_1 ))
```

```
## Errores con 10 iteraciones 29
```

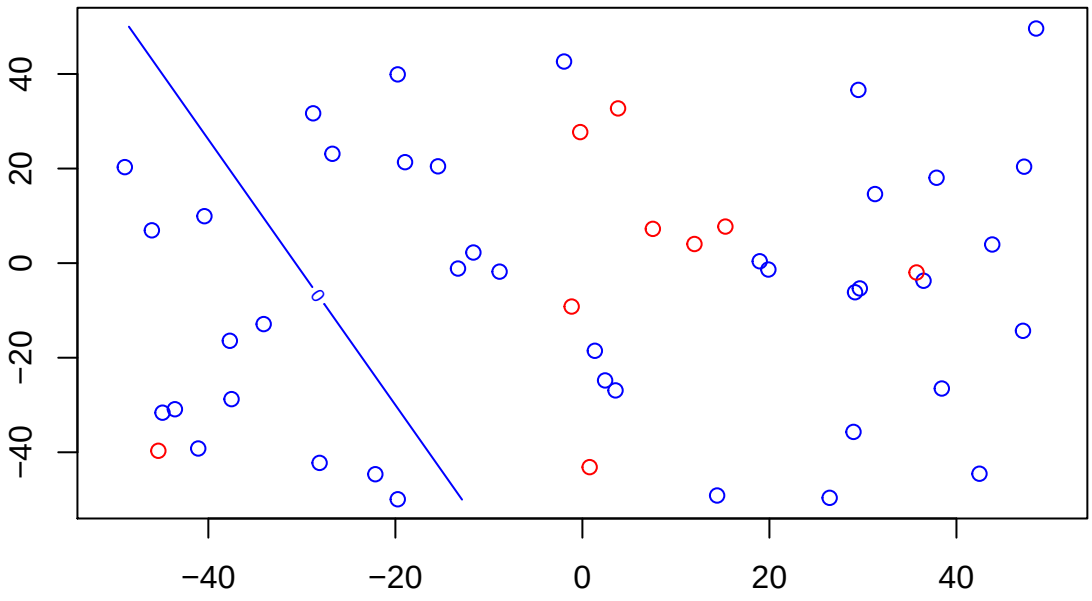
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```




```
sol <- ajusta_PLA(lista_unif,etiqueta_mod_1, 100,rep(0,3))$hiperplane
cat("Errores con 100 iteraciones", count_errors(hiperplane_to_function(sol),
                                               lista_unif, etiqueta_mod_1 ))
```

```
## Errores con 100 iteraciones 20
```

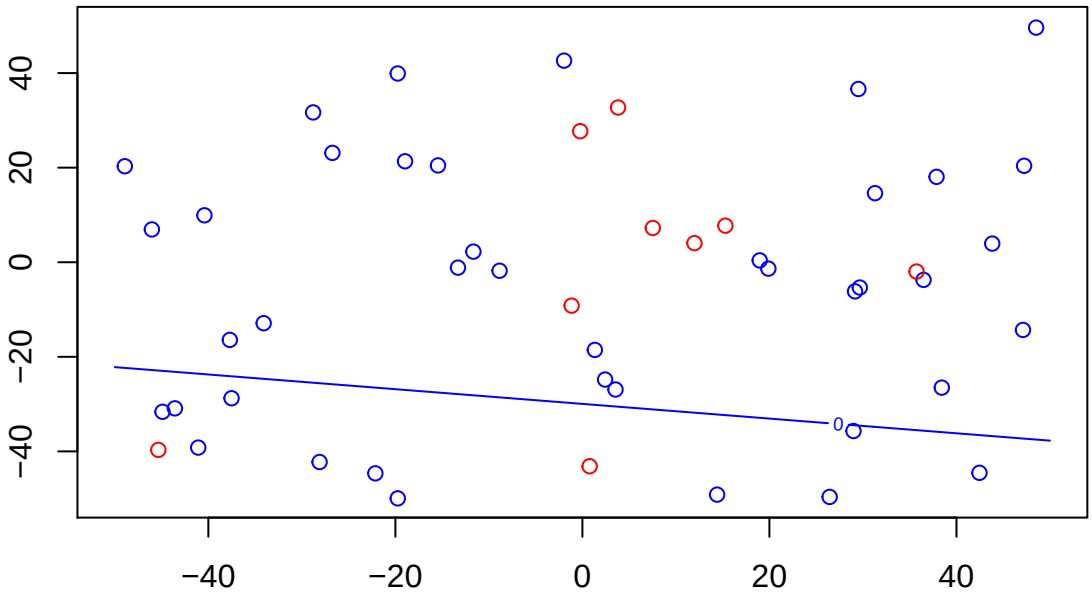
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```




```
sol <- ajusta_PLA(lista_unif,etiqueta_mod_1, 1000,rep(0,3))$hiperplane
cat("Errores con 1000 iteraciones", count_errors(hiperplane_to_function(sol),
                                        lista_unif, etiqueta_mod_1 ))
```

```
## Errores con 1000 iteraciones 18
```

```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```



De nuevo tenemos el problema de que los datos no son separables linealmente sin realizar una transformación, por lo que el resultado es muy malo, prácticamente aleatorio. Podemos, eso sí, al menos quedarnos con la mejor iteración del algoritmo.

Modifique la función `ajusta_PLA` para que le permita visualizar los datos y soluciones que va encontrando a lo largo de las iteraciones. Ejecute con la nueva versión el apartado 3 del ejercicio 5.3.

```
# ajusta_PLA(lista_unif,etiqueta, 1000,rep(0,3), TRUE)$hiperplane
# ajusta_PLA(lista_unif,etiqueta_mod, 1000,rep(0,3), TRUE)$hiperplane
```

A la vista de la conducta de las soluciones observada en el apartado anterior, proponga e implemente una modificación de la función original sol = `ajusta_PLA_MOD()` que permita obtener soluciones razonables sobre datos no linealmente separables. Mostrar y valorar el resultado encontrado usando los datos del apartado 7 del ejercicio 5.2

Como se ha mencionado anteriormente, el algoritmo implementado anteriormente no asegura ni siquiera que devolvamos la mejor solución encontrada, así que es lógico guardar siempre la mejor solución.

```
ajusta_PLA_MOD <- function(datos, label, max_iter, vini, draw_iterations = FALSE){
  sol <- vini
  iter <- 0
  changed <- TRUE
  current_sol <- sol

  while ( iter < max_iter && changed){
    current_errors <- count_errors(hiperplane_to_function(sol), datos, label )
    changed <- FALSE

    for( inner_iter in 1:ncol(datos) ){
      x <- c(datos[inner_iter,],1)

      if( sign(crossprod(x,current_sol)) != label[inner_iter]){
        current_sol <- current_sol + label[inner_iter] * x
        changed <- TRUE
      }
    }

    if(draw_iterations){
      draw_function(c(-50,50), c(-50,50),
                    function(x,y) y +sol[1]/sol[2]*x +sol[3]/sol[2], col = 4)
      draw_function(c(-50,50), c(-50,50),
                    function(x,y) y +current_sol[1]/current_sol[2]*x
                    +current_sol[3]/current_sol[2], col = 5, add=TRUE)
      points(lista_unif[,1], lista_unif[,2], col = (label+3))
      Sys.pause(0.1)
    }

    if( current_errors >= count_errors(hiperplane_to_function(current_sol),
                                       datos, label )){
      sol <- current_sol
    }
```

```
    iter <- iter+1
  }
  sol <- sol/sol[length(sol)-1]

  return( list( hiperplane = sol, iterations = iter))
}
```

Para ver cómo afecta esto a la ejecución, mostramos las iteraciones para el caso del ejercicio anterior.

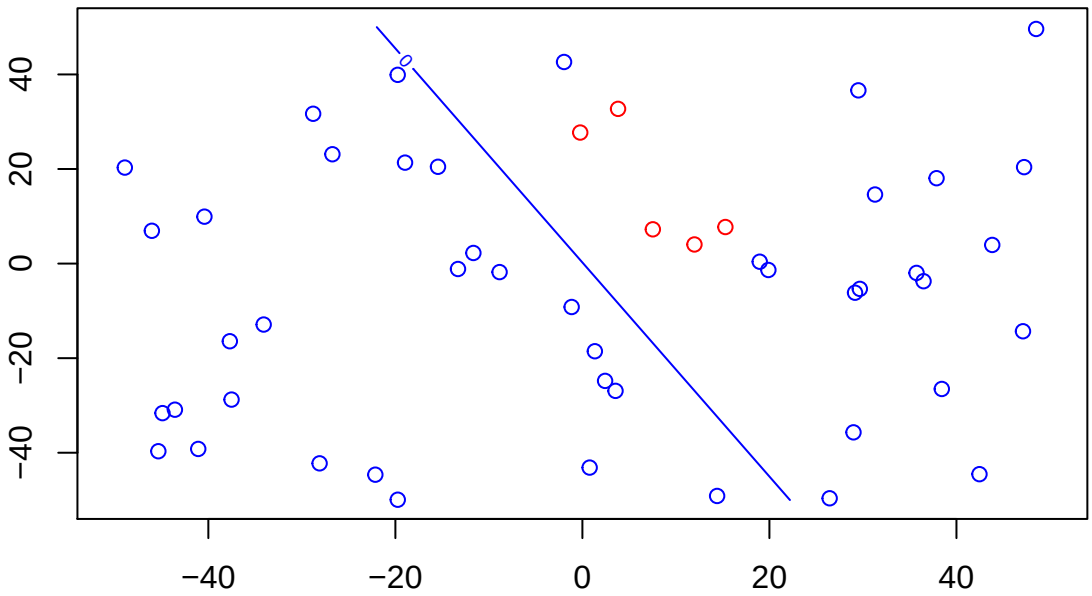
```
#ajusta_PLA(lista_unif,etiqueta_mod, 1000,rep(0,3), TRUE)$hiperplane
```

Lo lanzamos ahora con las funciones cuadráticas. y mostramos el número de errores:

```
sol_cuadratic_1 <- ajusta_PLA_MOD(lista_unif, etiqueta_1, 1000, rep(0,3))$hiperplane
cat("Errores f_1", count_errors(hiperplane_to_function(sol_cuadratic_1),
                                lista_unif, etiqueta_1 ))
```

```
## Errores f_1 32
```

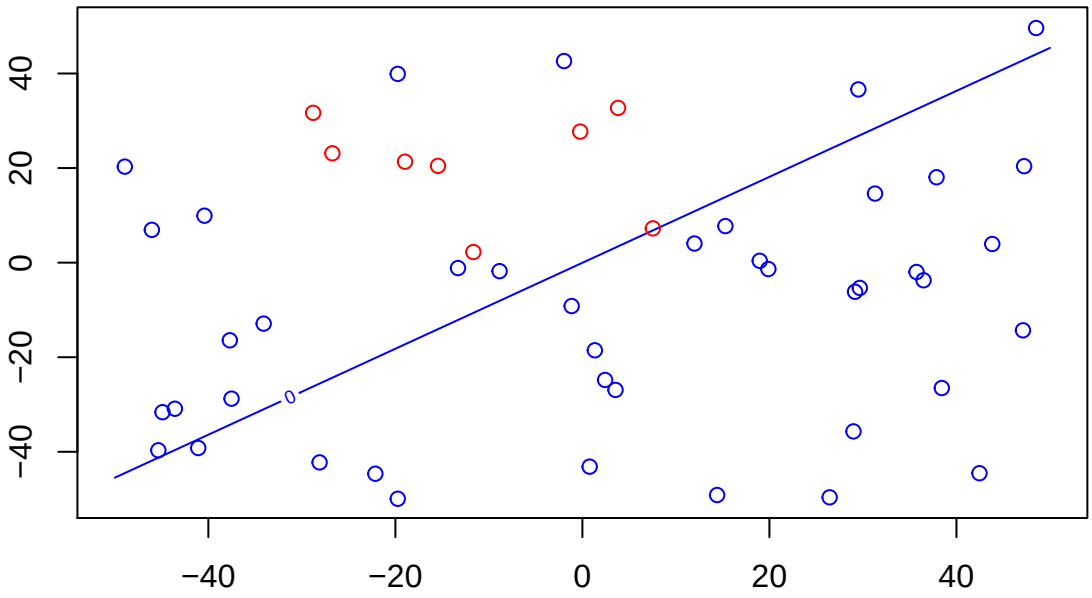
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_1[1]*x +sol_cuadratic_1[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_1+3))
```




```
sol_cuadratic_2 <- ajusta_PLA_MOD(lista_unif, etiqueta_2, 1000, rep(0,3))$hiperplane
cat("Errores f_2", count_errors(hiperplane_to_function(sol_cuadratic_2),
                                lista_unif, etiqueta_2 ))
```

```
## Errores f_2 35
```

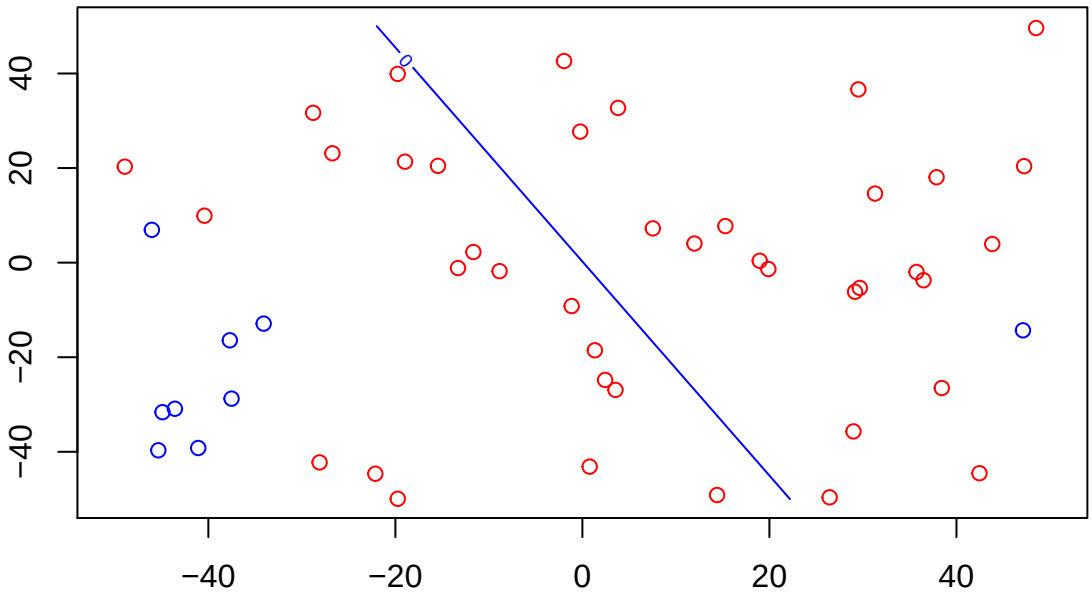
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_2[1]*x +sol_cuadratic_2[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_2+3))
```




```
sol_cuadratic_3 <- ajusta_PLA_MOD(lista_unif, etiqueta_3, 1000, rep(0,3))$hiperplane
cat("Errores f_3", count_errors(hiperplane_to_function(sol_cuadratic_3),
                                lista_unif, etiqueta_3 ))
```

```
## Errores f_3 30
```

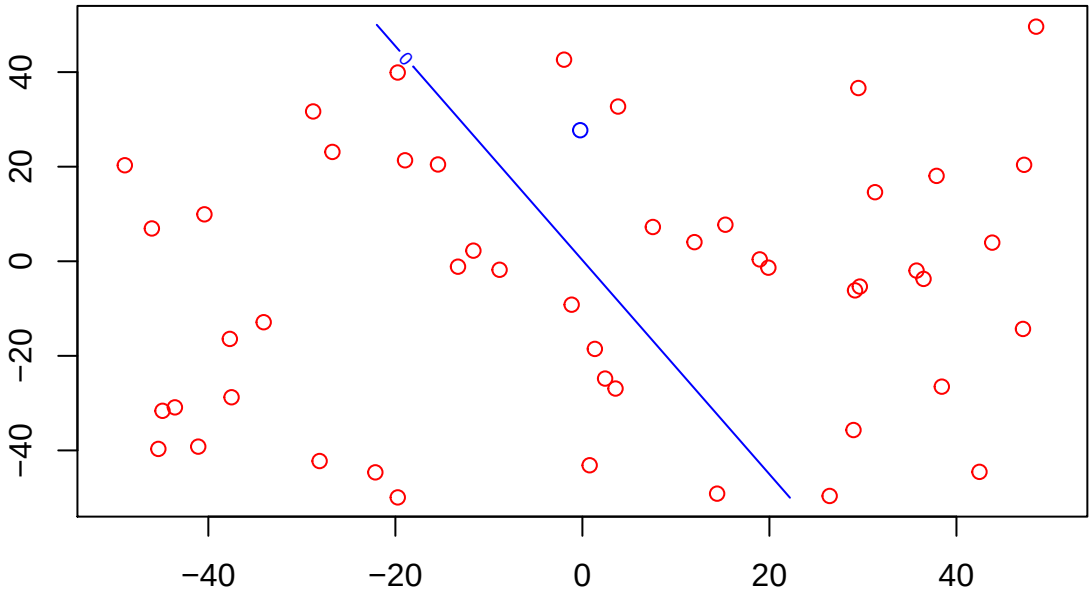
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_3[1]*x +sol_cuadratic_3[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_3+3))
```



```
sol_cuadratic_4 <- ajusta_PLA_MOD(lista_unif, etiqueta_4, 1000, rep(0,3))$hiperplane
cat("Errores f_4", count_errors(hiperplane_to_function(sol_cuadratic_4),
                                lista_unif, etiqueta_4 ))
```

```
## Errores f_4 22
```

```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_4[1]*x +sol_cuadratic_4[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_4+3))
```



Regresión lineal

Lectura de datos y dotación de forma

```
datos = scan("datos/zip.train",sep=" ")
datos = matrix(datos,ncol=257,byrow=T)
datos <- datos[-(which(datos[,1] != 1 & datos[,1]!=5)),]

number <- datos[,1]
pixels <- aperm(array(t(datos[,2:257]), dim=c(16,16,nrow(datos))), perm=c(2,1,3))
```

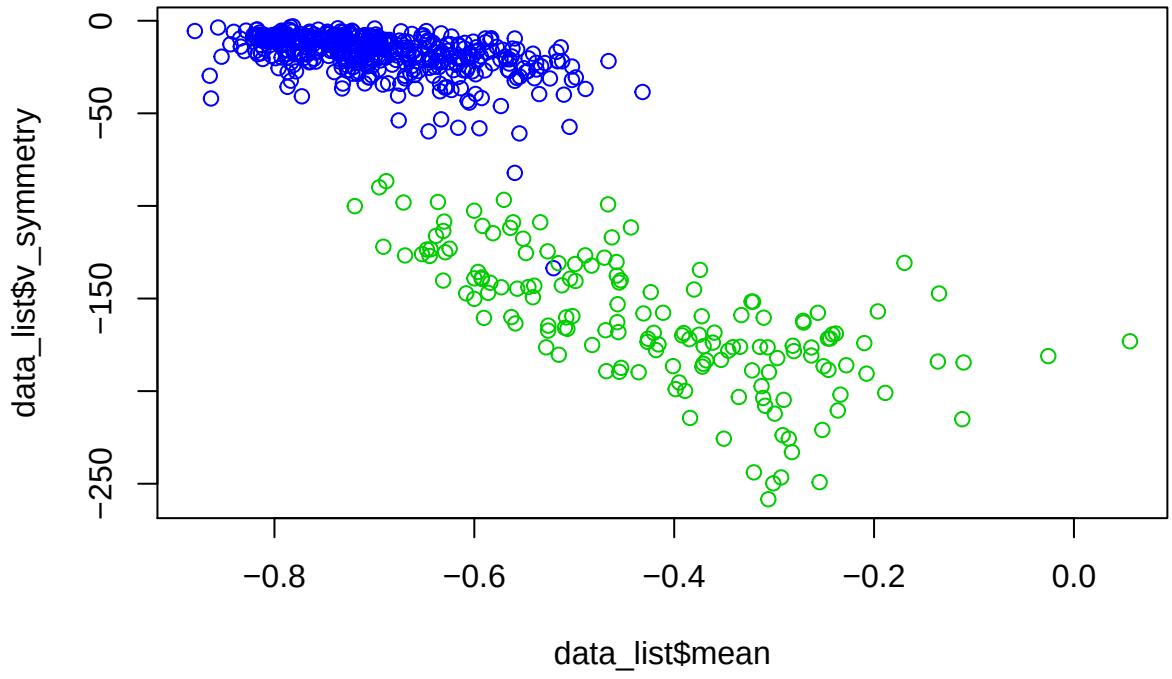
Cálculo de media y simetría

```
means <- apply(pixels, 3, mean)

simetria_vertical <- function(M){
  -sum(apply(M, 2, function(x) sum(abs(x-x[length(x):1]))))
}
```

```
sim_vertical <- apply(pixels,3,simetria_vertical)
```

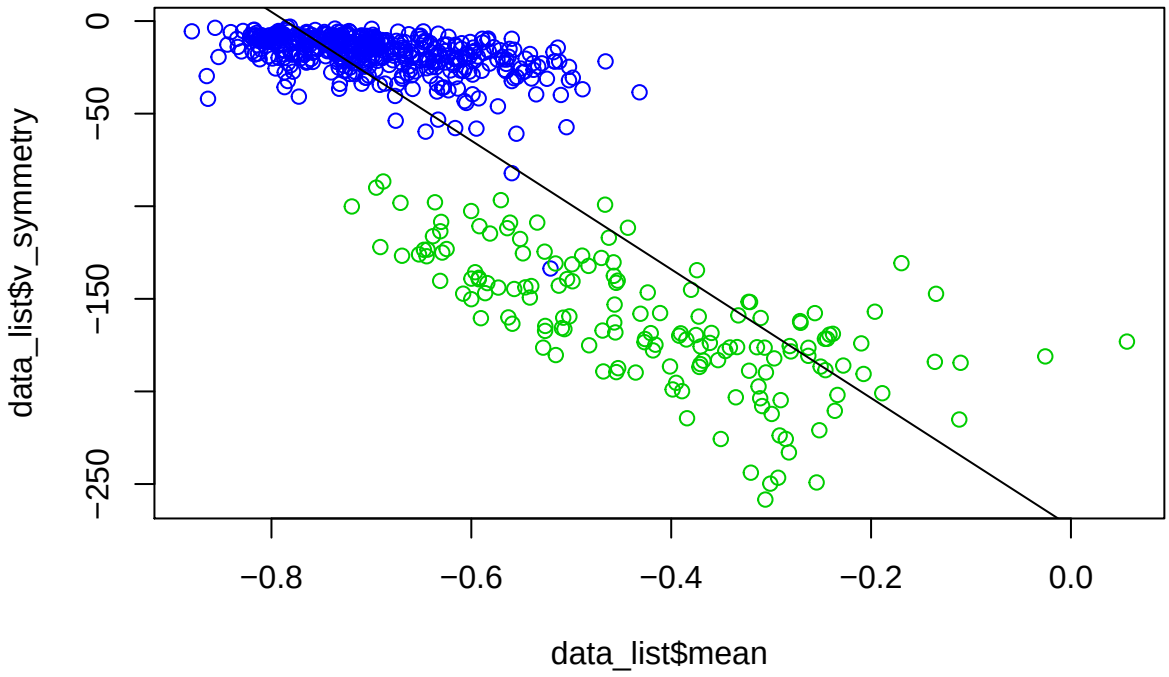
Representación

```
number_colors = numeric(length(number))
number_colors[which(number==1)] <- 4
number_colors[which(number==5)] <- 3
data_list <- list('number'=number, 'pixels'=pixels, 'mean'=means,
                  'v_symmetry'=sim_vertical, 'colors'=number_colors )
plot(data_list$mean, data_list$v_symmetry, col = data_list$colors )
```



Implementar la función `sol = Regress_Lin(datos, label)` que permita ajustar un modelo de regresion lineal (usar SVD). Los datos de entrada se interpretan igual que en clasificación.

```
regress_lin <- function(datos, label){
  desc <- svd(datos)
  d <- round(desc$d, digits=5)
  d[abs(d)>0] <- 1/d[abs(d)>0]
  pseudo_inv = desc$v %*% diag(d) %*% t(desc$u)
  return(pseudo_inv%*%label)
}
plot(data_list$mean, data_list$v_symmetry, col = data_list$colors )
coefs <- regress_lin(cbind(means,rep(1,length(means))), sim_vertical)
abline(coefs[2],coefs[1])
```



En este ejercicio exploramos cómo funciona regresión lineal en problemas de clasificación. Para ello generamos datos usando el mismo procedimiento que en ejercicios anteriores. Suponemos $X = [10, 10] \times [10, 10]$ y elegimos muestras aleatorias uniformes dentro de X . La función f en cada caso será una recta aleatoria que corta a X y que asigna etiqueta a cada punto con el valor de su signo. En cada apartado generamos una muestra y le asignamos etiqueta con la función f generada. En cada ejecución generamos una nueva función f

Fijar el tamaño de muestra $N = 100$. Usar regresión lineal para encontrar g y evaluar E_{in} , (el porcentaje de puntos incorrectamente clasificados). Repetir el experimento 1000 veces y promediar los resultados ¿Qué valor obtiene para E_{in} ?