Práctica 1 Aprendizaje Automático

Jacinto Carrasco Castillo

A - 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()
}
```

1. 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.

Dado que es lo que indica el enunciado y aunque en el desarrollo de la práctica se da a entender que esta función podría requerir distintos intervalos para cada función, no se pide, sólo he incluido un vector *rango* que será el intervalo en el que estén los puntos para todas las dimensiones.

```
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)
}
```

Situamos los datos por filas para una más fácil visualización. Tendremos por tanto una matriz con tantas filas como el número de datos y tantas columnas como dimensiones.

2. 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)
}
```

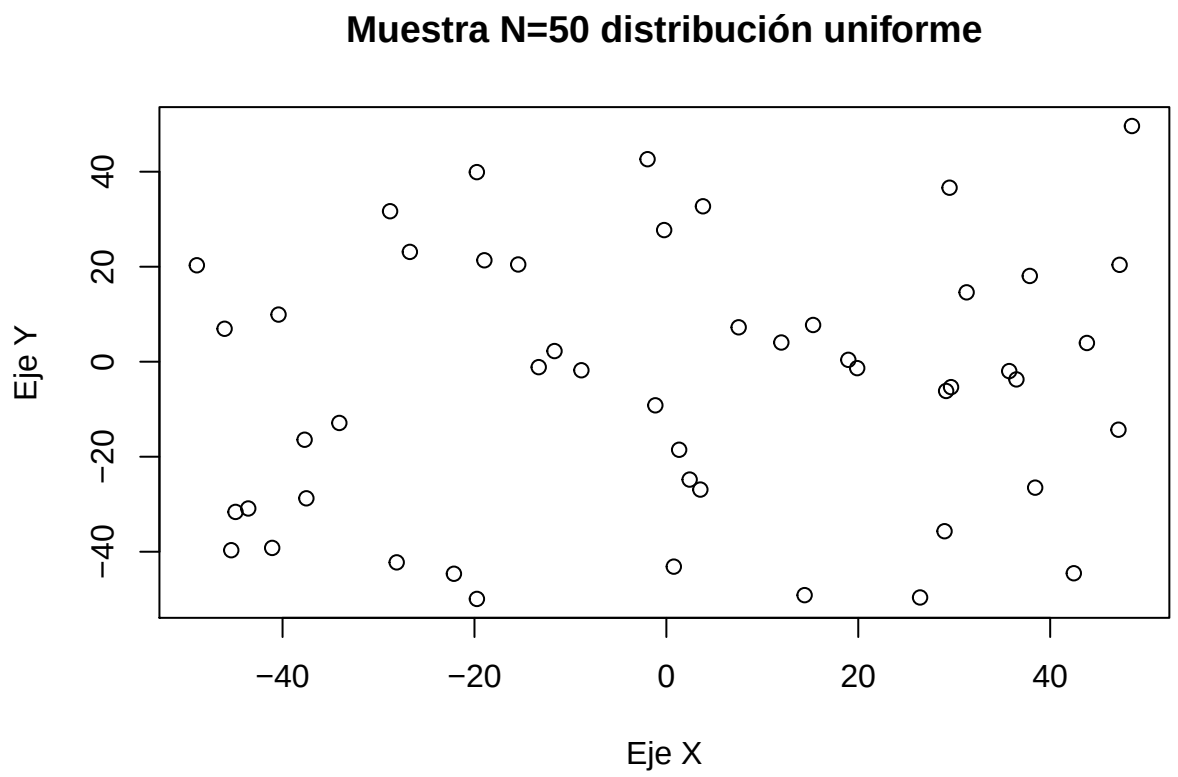
En este caso estamos rellenando la matriz por filas. Haciéndolo de esta manera, y puesto que sigma es un vector cuya dimensión debería ser 1 o coincidir con la dimensión de los vectores (dim), el número generado se genera con la desviación típica correspondiente a la de su dimensión, y se irá repitiendo hasta cubrir el número de datos a generar ($N \cdot dim$) con lo que lo hará para cada fila (dato).

3 y 4. 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.

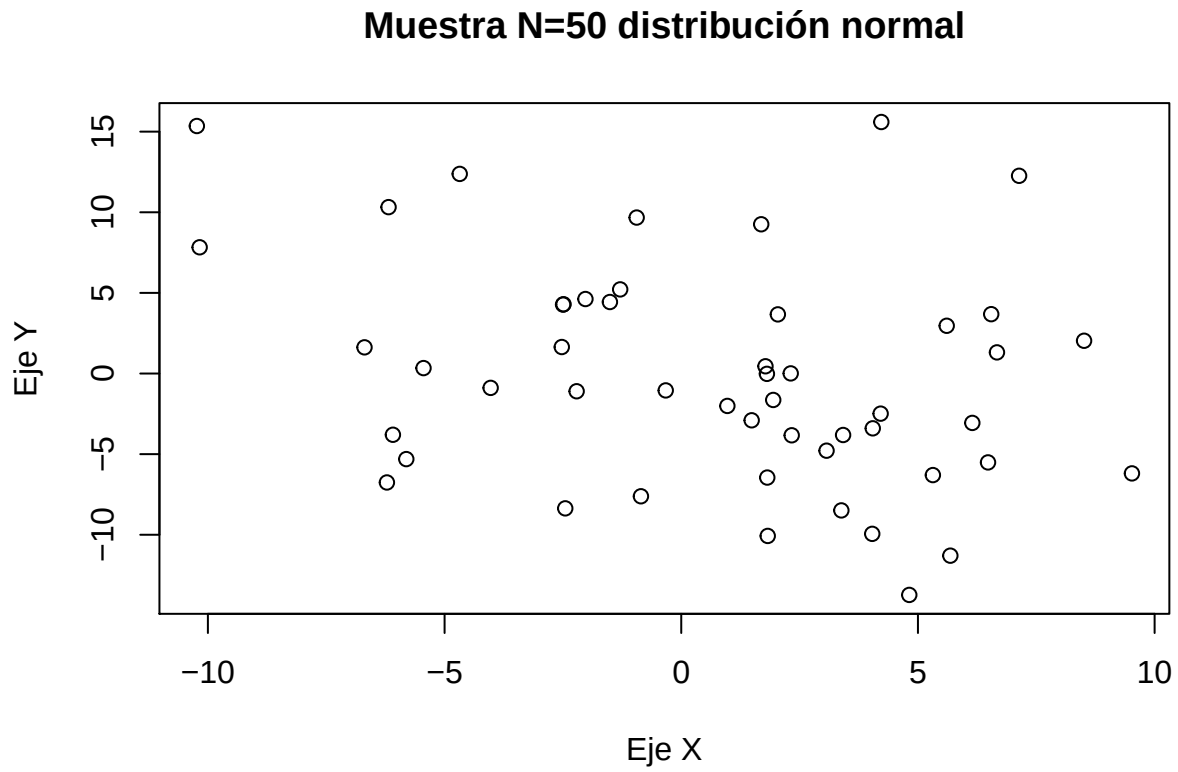
```
lista_unif <- simula_unif(50, 2, c(-50, 50))
lista_gauss <- simula_gauss(50, 2, c(5,7))
```

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, se aprecia una mayor concentración de puntos en torno al centro en la distribución normal, siendo menos frecuentes por las esquinas. En cambio en la distribución uniforme los datos están, como era de esperar, más repartidos.

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

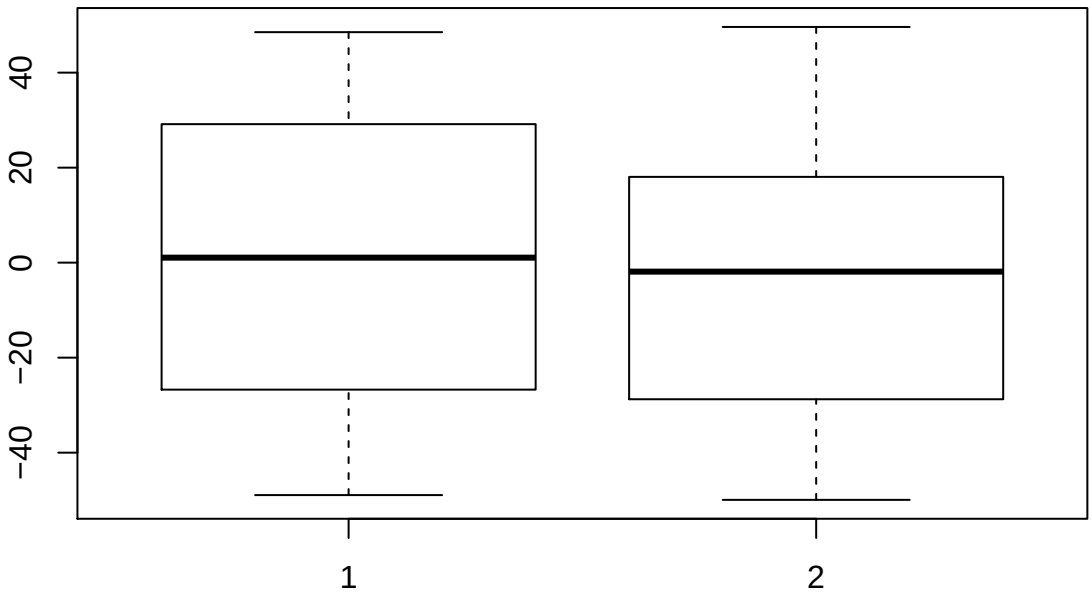



```
plot(lista_gauss[,1],lista_gauss[,2], xlab="Eje X", ylab="Eje Y",
     main="Muestra N=50 distribución normal")
```

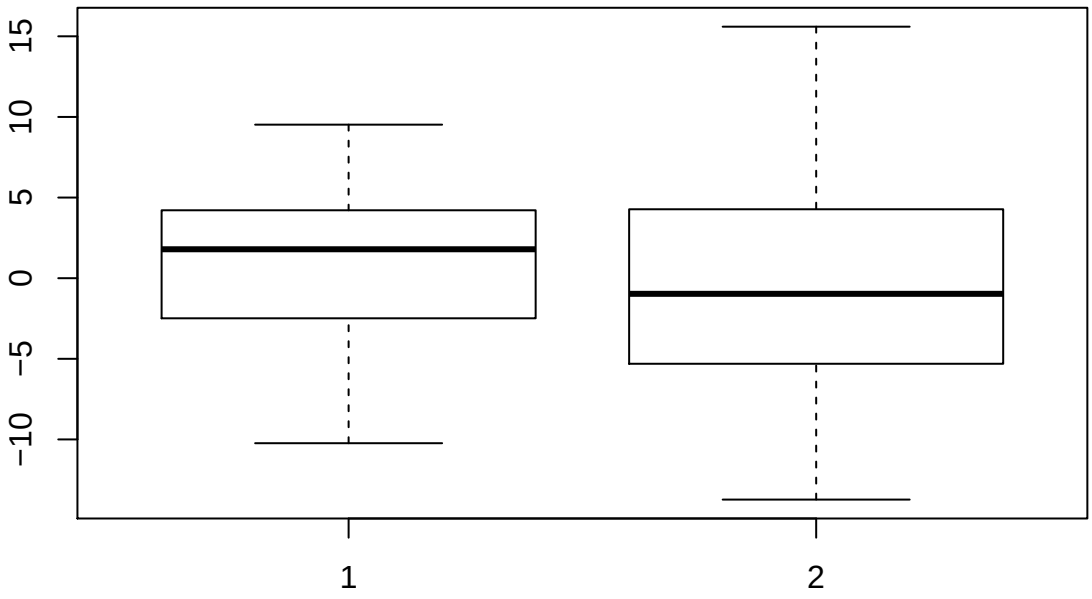


Si mostramos ahora los diagramas de cajas con bigote para cada una de las dimensiones tanto en la distribución uniforme como en la distribución de Gauss, vemos que en la distribución uniforme ambos diagramas son muy parecidos (como era de esperar pues la distribución es la misma), mientras que en la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

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




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



5. 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. La función `v=simula_recta(intervalo)` devolverá una función en $\mathbb{R}^2$ que represente la recta generada.

```
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]

  f <- function(x,y){
    return(y - a*x -b)
  }
  return(f)
}
```

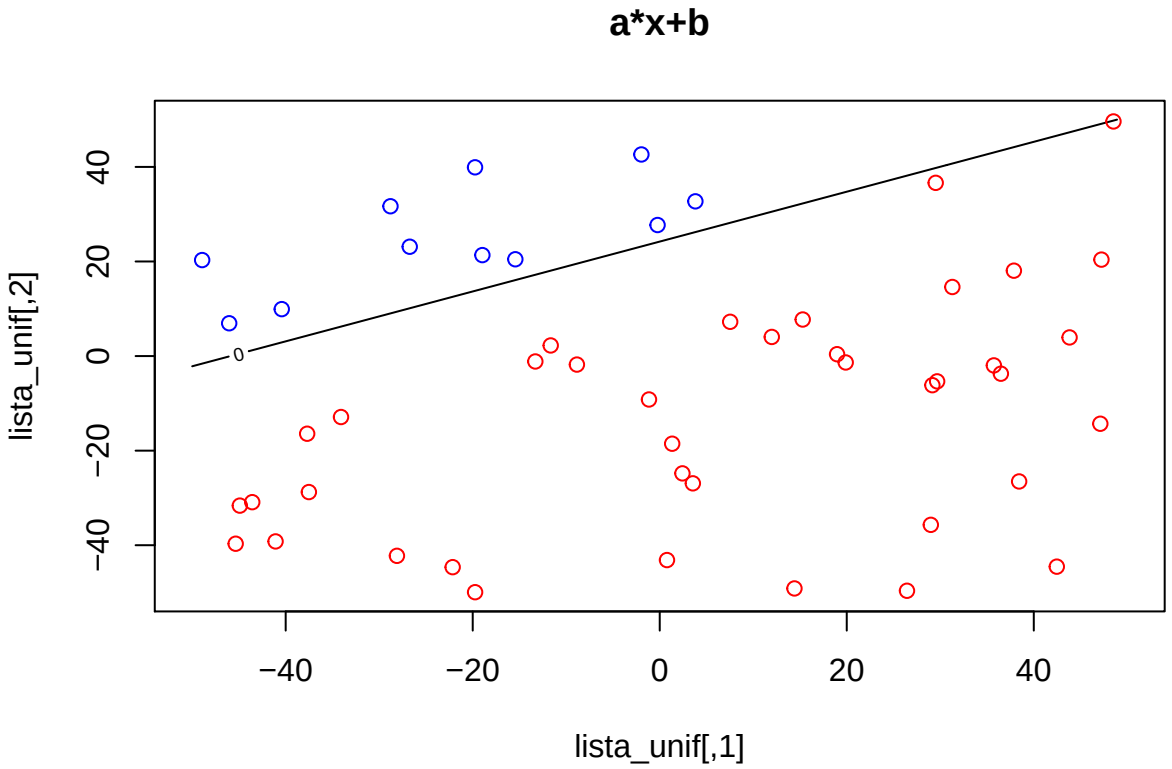
6. Generar una muestra 2D de puntos usando `simula_unif()` y etiquetar la muestra usando el signo de la función $f(x, y) = y - ax - b$ de cada punto a una recta simulada con `simula_recta()`. Mostrar una gráfica con el resultado de la muestra etiquetada junto con la recta usada para ello.

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

```
intervalo <- c(-50, 50)
f <- simula_recta(intervalo)
etiqueta <- apply(lista_unif, 1, function(X) sign(f(X[1],X[2])))
```

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), f)
title(main="a*x+b", ylab = "lista_unif[,2]", xlab = "lista_unif[,1]")
points(lista_unif[,1], lista_unif[,2], col = (etiqueta+3))
```

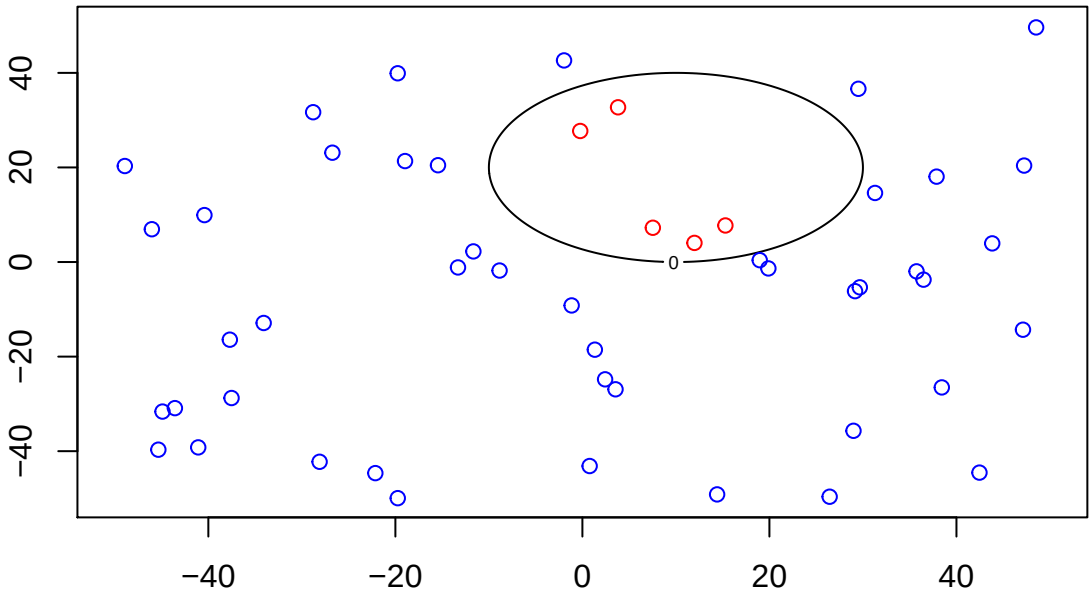


Como vemos en la gráfica, los puntos rojos (etiquetados con -1) se sitúan debajo de la gráfica y los azules por encima. Para hacer esto, aunque se podría generar un vector de colores en función de las etiquetas, es más cómodo sumar 3 al vector de etiquetas para obtener así $(2, 4)$ y que los colores sean rojo y azul, respectivamente, que se distinguen bien y se realiza la asociación de rojo con estar por debajo de cero.

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

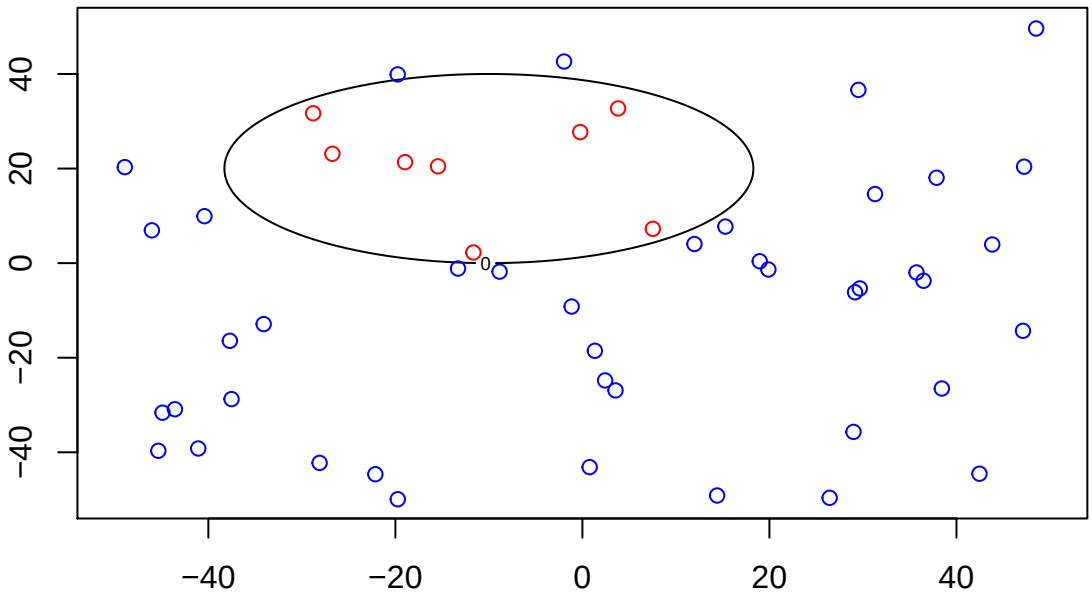
Aquí es donde verdaderamente entra en juego la función `contour`, pues nos permite representar una función cuadrática de manera sencilla. Para realizar el etiquetado, lo hacemos como para el caso de la recta, tomando el signo de la evaluación de la función en cada dato de la muestra. En este caso la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



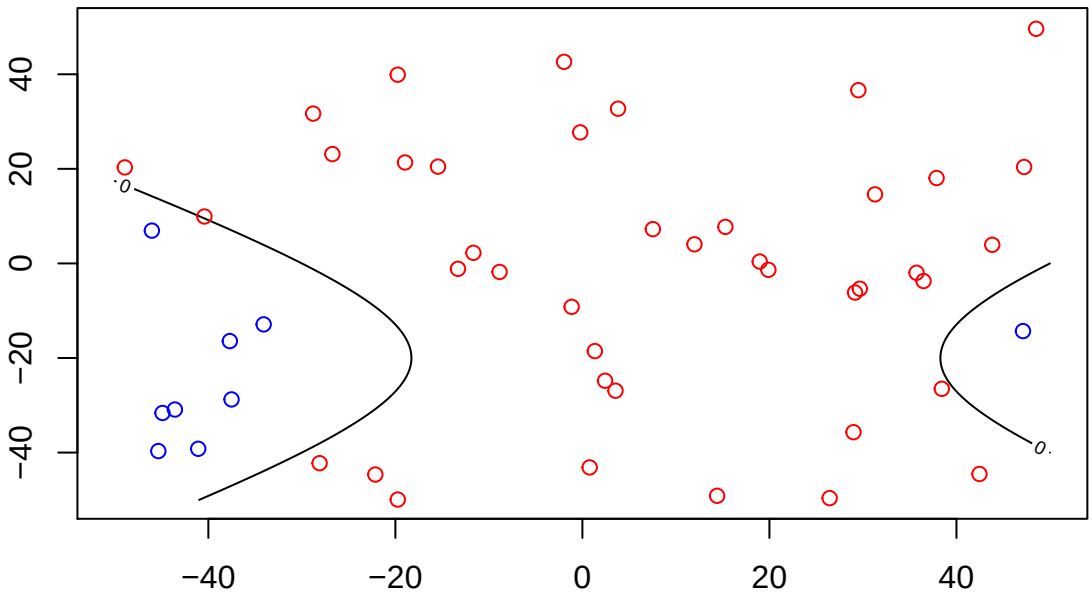
Para esta segunda función, también la mayoría de datos cae fuera de la cuádrica y es una elipse, como podíamos deducir de sus coeficientes.

```
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))
```



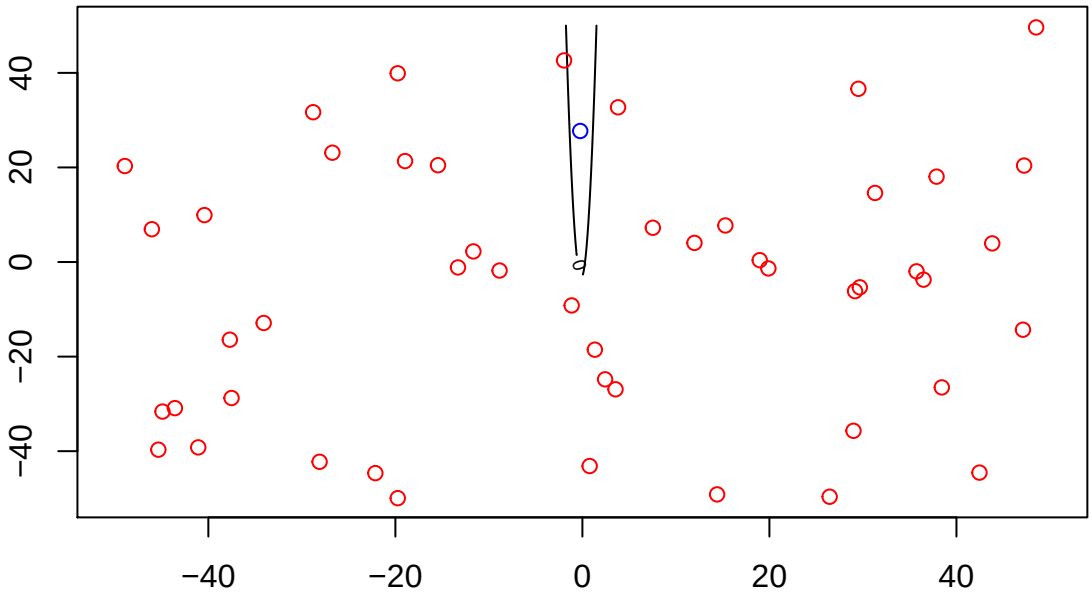
En esta hipérbola, la mayoría de los datos caen fuera, o bien en la parte izquierda.

```
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.

```r
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 sin hacer antes una transformación, y será necesario una nueva aproximación para resolver el problema.

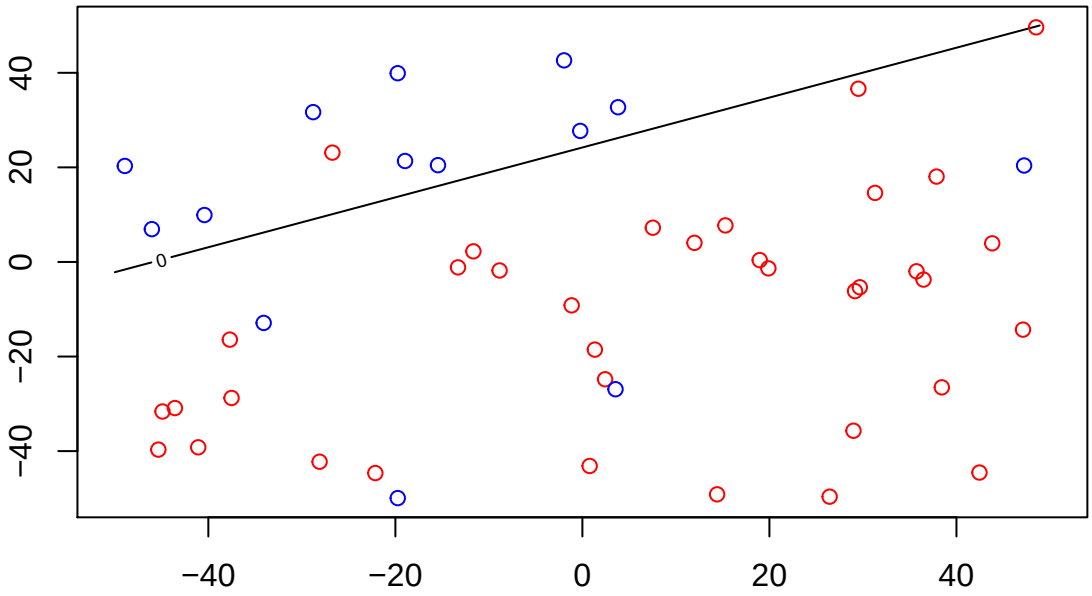
8. Considerar de nuevo la muestra etiquetada en el apartado 6. Modifique las etiquetas de un 10 aleatorio de muestras positivas y otro 10 aleatorio de negativas.

```r
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
  }

  #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
```

```
  }

  return(mod_v)
}
```

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

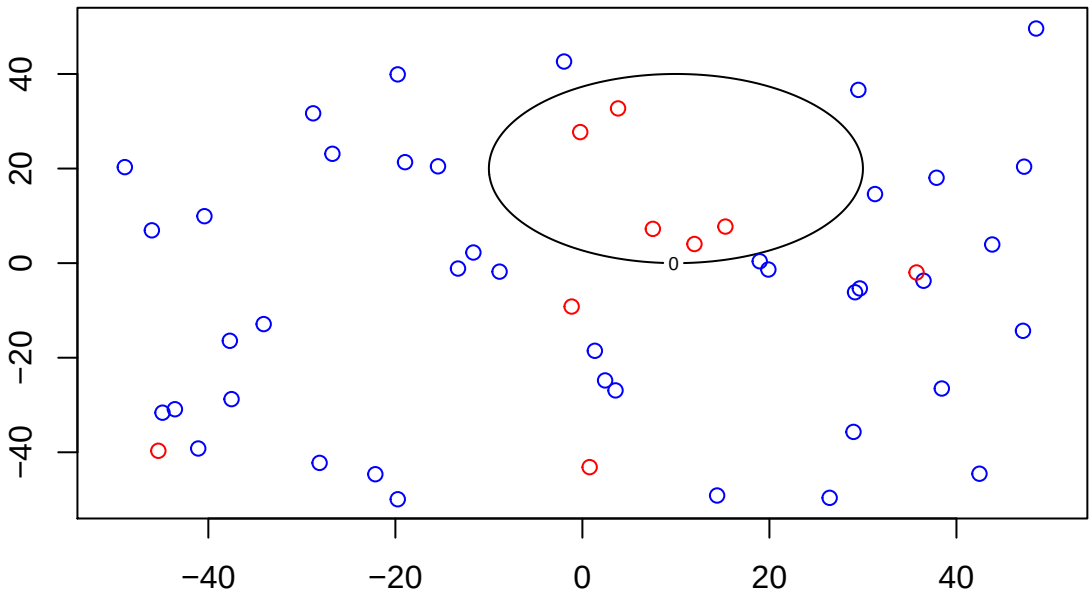
```
etiqueta_mod <- modify_rnd_bool_subvector(etiqueta)

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



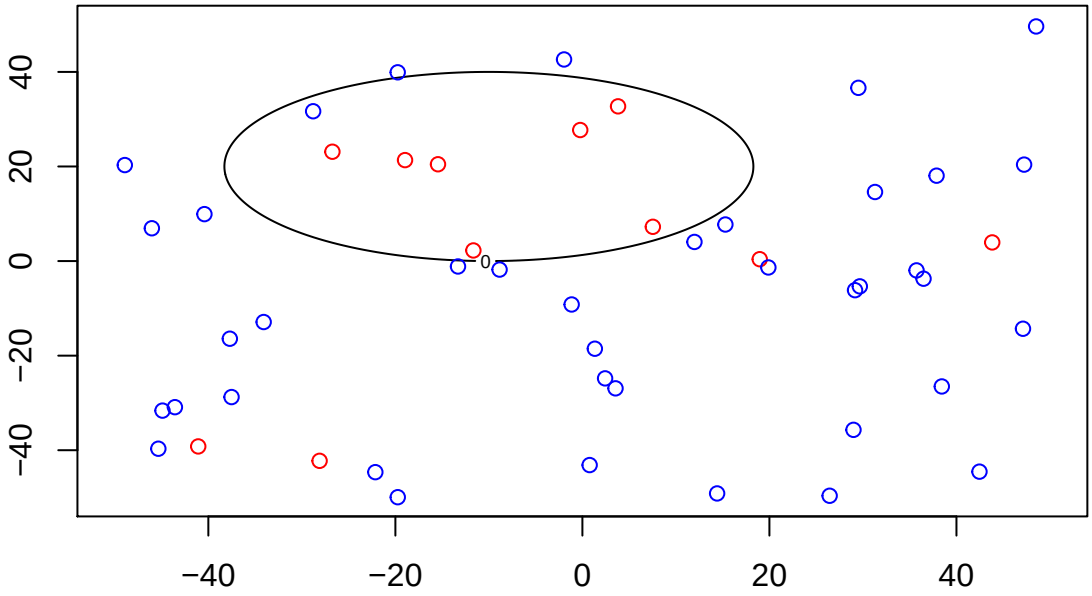
Ahora hay datos que podríamos considerar mal clasificados, pues se sitúan en el lado erróneo de la recta. Sin embargo, esto excede la capacidad de las herramientas vistas hasta ahora, ya que al no ser linearmente separables el Perceptron no dará con las solución correcta.

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

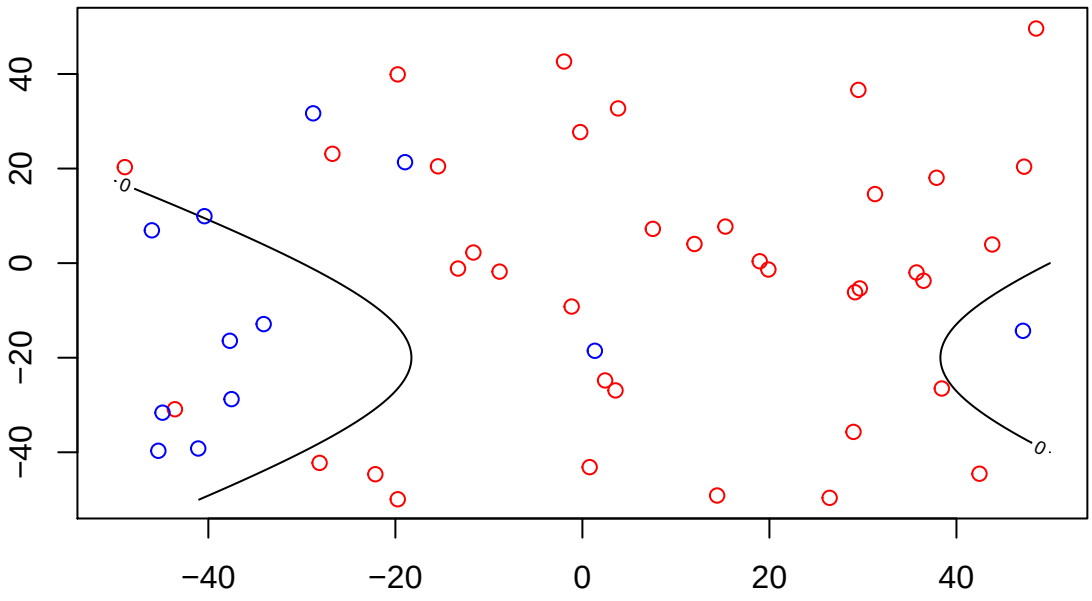
```
etiqueta_mod_1 <- modify_rnd_bool_subvector(etiqueta_1)

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



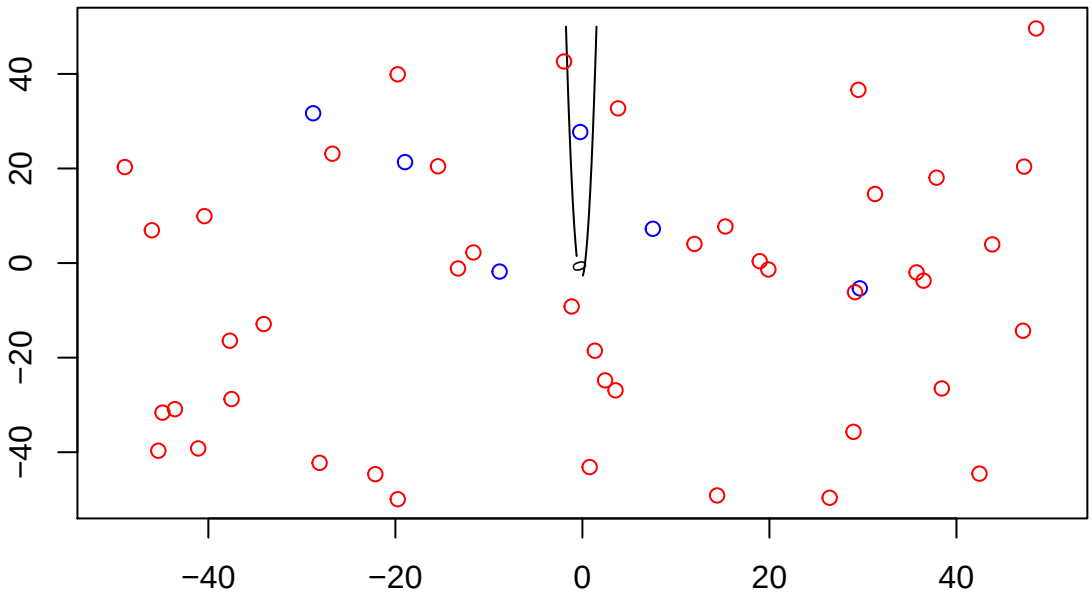

```
etiqueta_mod_2 <- modify_rnd_bool_subvector(etiqueta_2)

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.

B - Ajuste del algoritmo Perceptron

1. 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.

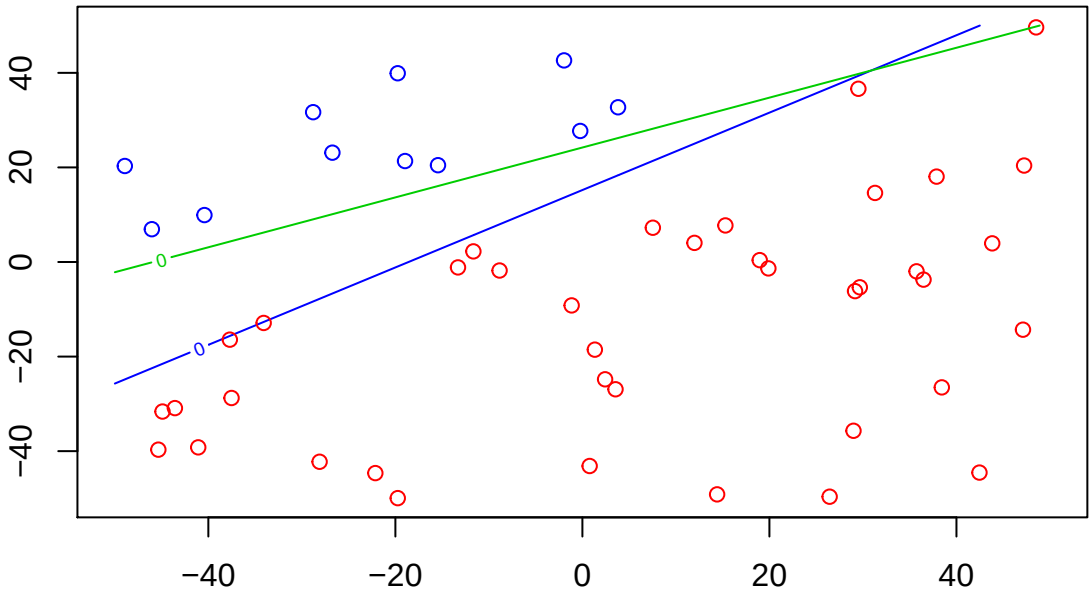
```
ajusta_PLA <- function(datos, label, max_iter, vini, draw_iterations = FALSE){
  sol <- vini
  iter <- 0
  changed <- TRUE

  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)

      #Modificamos la solución si encontramos un punto mal clasificado.
      if( sign(crossprod(x,sol)) != label[inner_iter]){
        sol <- sol + label[inner_iter] * x
        changed <- TRUE
      }
    }
    #Dibujamos cada iteración del algoritmo, haciendo una pausa para visualizarlo
    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 mostramos un ejemplo de la ejecución del algoritmo PLA, que da la solución en azul, mientras que se pinta en verde la recta con la que realizamos la clasificación. Se

aprecia que efectivamente se ha resuelto en, 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 sin conocer la solución real no tenemos 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))
draw_function(c(-50,50), c(-50,50), f, col = 3, add = TRUE)
```



2. 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(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 55 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, algo que se puede comprobar cambiando la semilla, la convergencia cuando ocurre esto es mucho

más rápida, 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.

3. 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
  signs <- apply(datos, 1, function(x) return(sign(f(c(x,1)))))
  return( sum(signs != label) )
}
```

Al haber hecho la función para contar errores de esta manera (porque tendremos funciones que no son lineales) necesitamos pasar de un vector (hiperplano) a una función lineal.

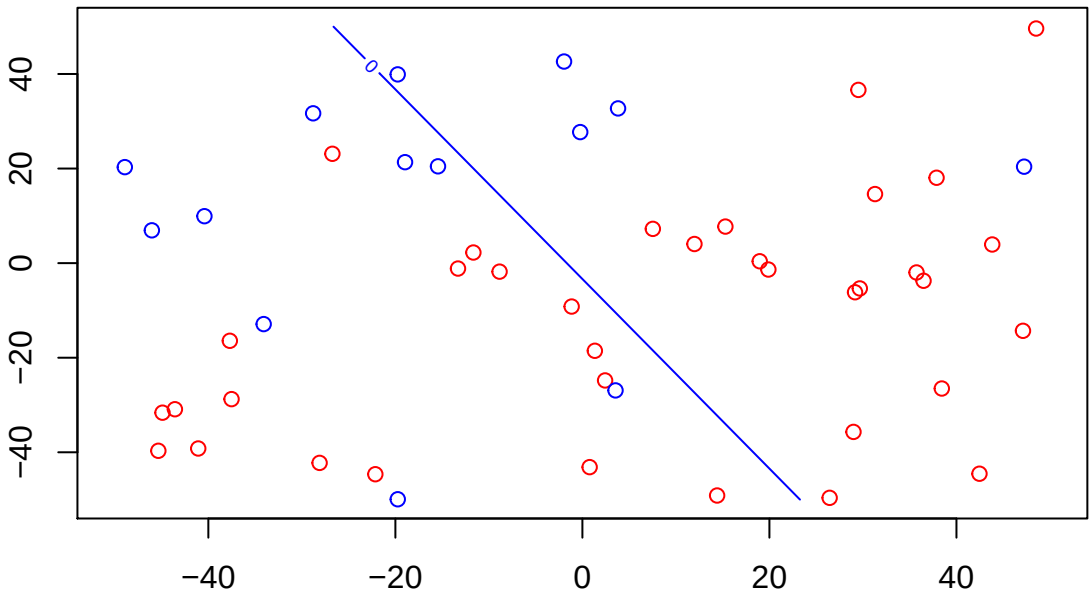
```
hiperplane_to_function <- function( vec ){
  f <- function(x){
    return( crossprod(vec,x) )
  }
  return(f)
}
```

Ahora para 10, 100 y 1000 iteraciones mostramos los errores cometidos:

```
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
```

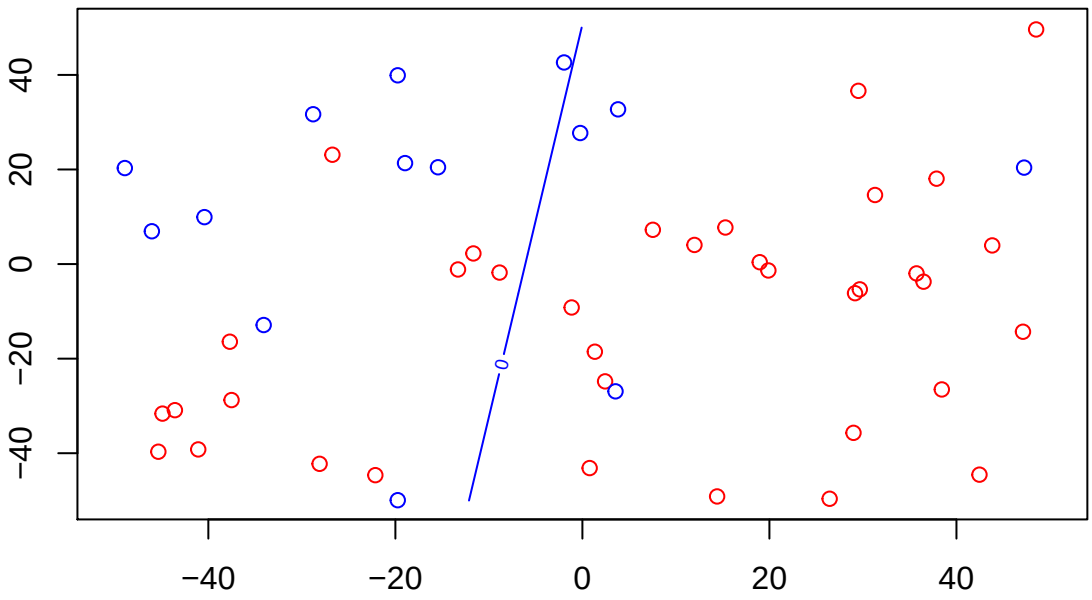
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```




```
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
```

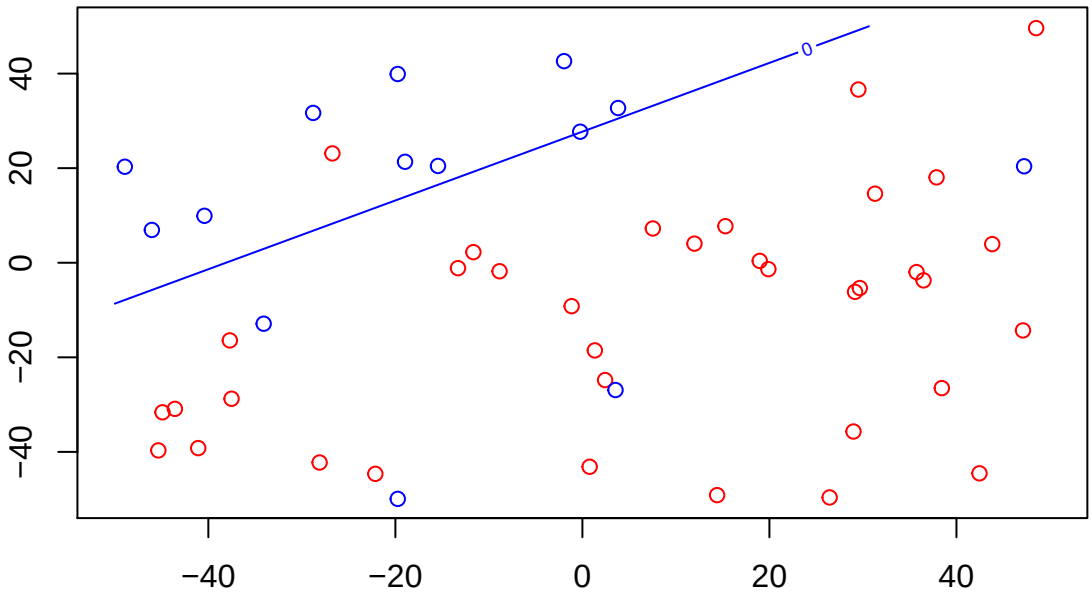
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod+3))
```




```
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))
```



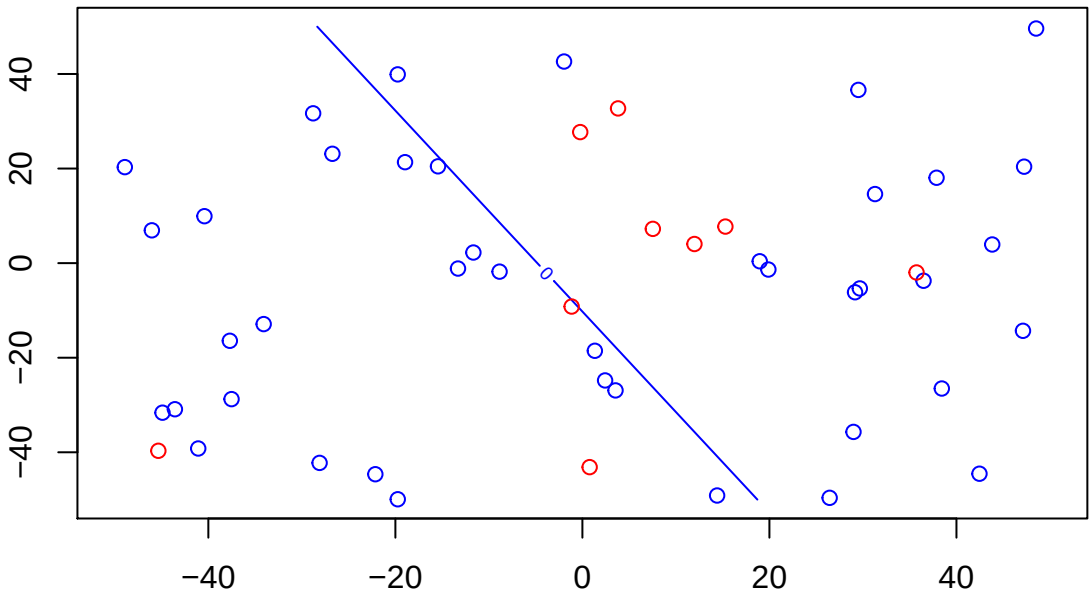
Ahora, con 1000 iteraciones llega a una solución equivalente a la del apartado anterior, aunque lógicamente con errores pues los datos no son separables. Aún así, es la mejor solución que podíamos encontrar ya que sólo están mal clasificados los datos cuya etiqueta hemos cambiado, y así se halla el mínimo 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.

4. 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
```

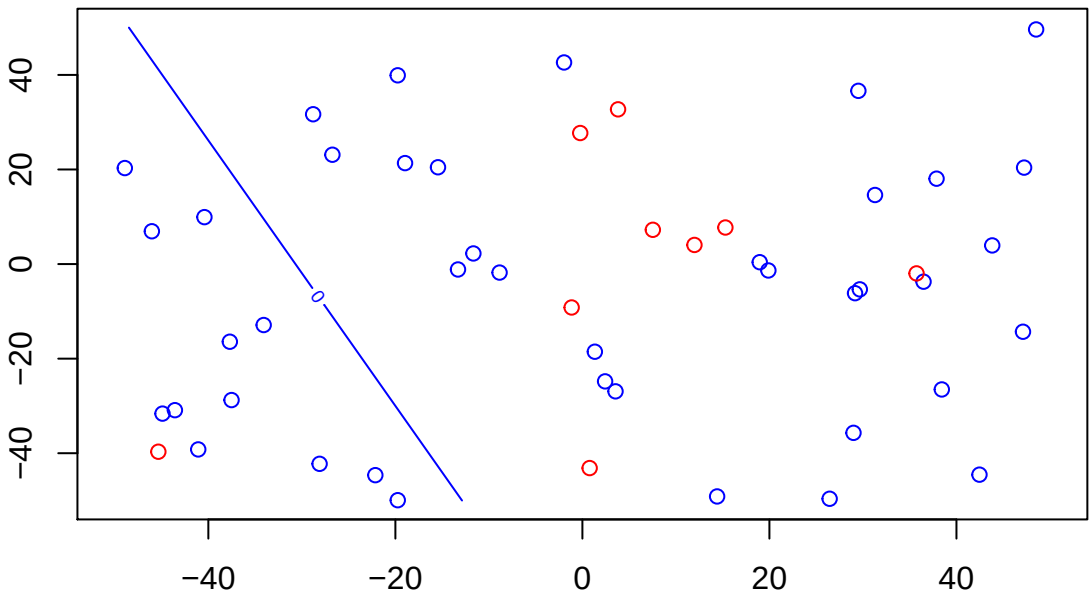
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```




```
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
```

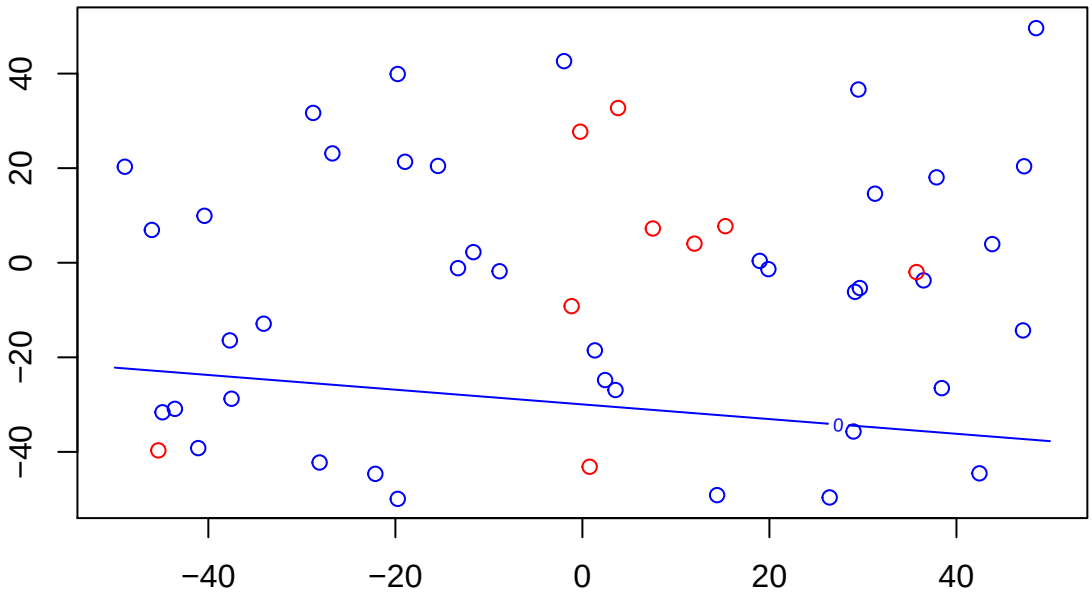
```
draw_function(c(-50,50), c(-50,50), function(x,y) y+sol[1]*x+sol[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_mod_1+3))
```




```
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, y además vemos como el algoritmo tiende a situar todos los puntos con etiqueta positiva por encima de la recta aunque tenga algunos puntos con etiqueta negativa mal clasificados. Podemos, al menos quedarnos con la mejor iteración del algoritmo.

5. 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
```

6. 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){
    # Comprobaremos en cada vuelta si hemos hecho algún cambio
```

```
    current_errors <- count_errors(hiperplane_to_function(sol), datos, label )
    changed <- FALSE

    for( inner_iter in 1:nrow(datos) ){
      # Añadimos coeficiente independiente
      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[2] +sol[1]*x +sol[3], col = 4)
      draw_function(c(-50,50), c(-50,50),
                    function(x,y) y*current_sol[2] +current_sol[1]*x +current_sol[3],
                    col = 5, add=TRUE)
      points(lista_unif[,1], lista_unif[,2], col = (label+3))
      Sys.sleep(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.

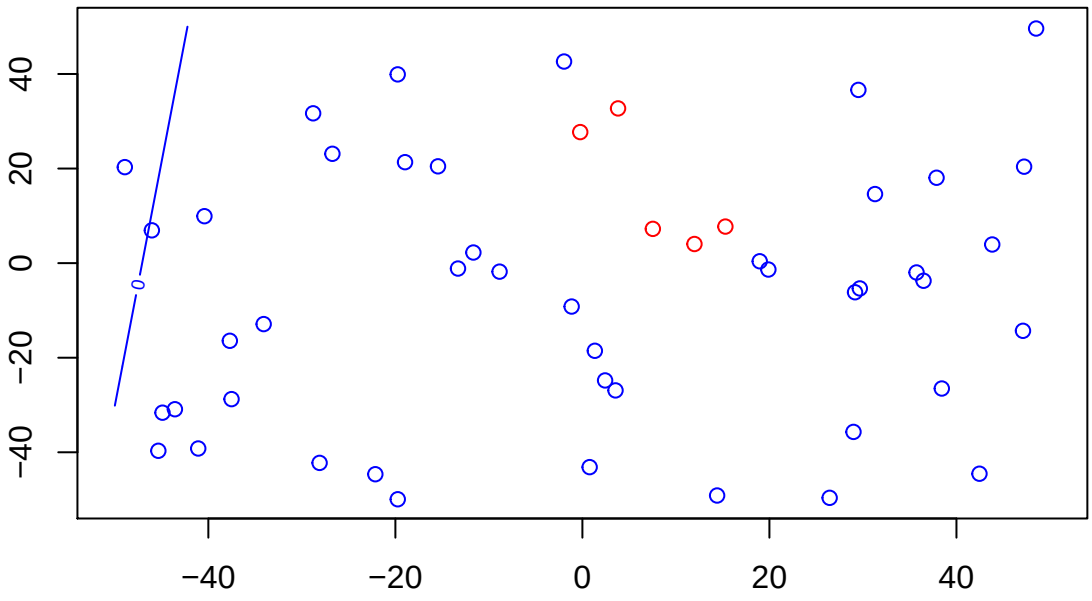
```
ajusta_PLA_MOD(lista_unif,etiqueta_mod, 1000,rep(0,3), TRUE)$hiperplane
```

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 44
```

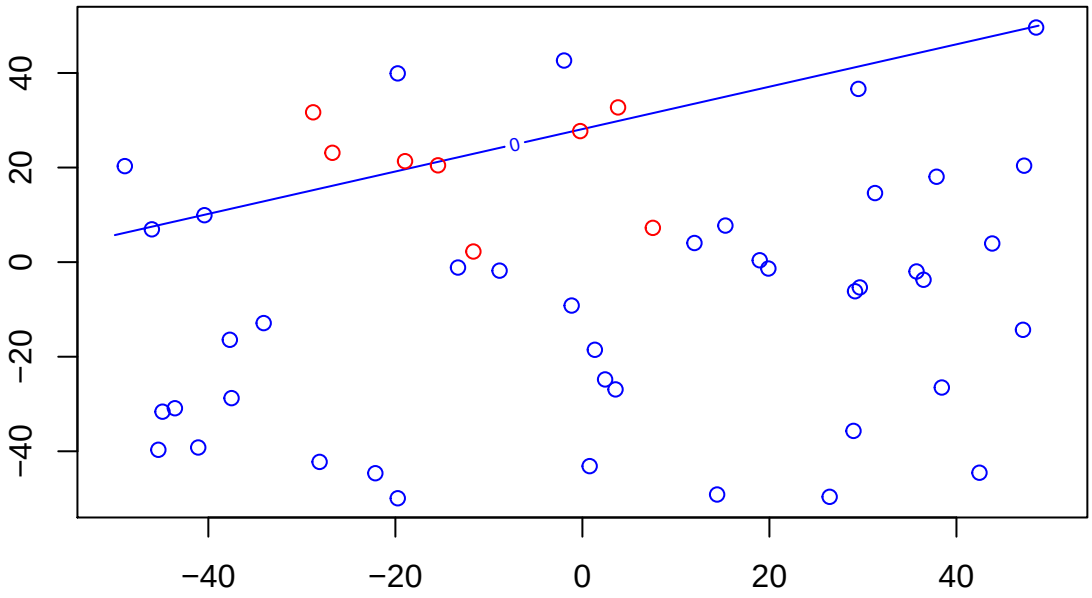
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_1[1]*x +sol_cuadratic_1[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_1+3))
```




```
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 43
```

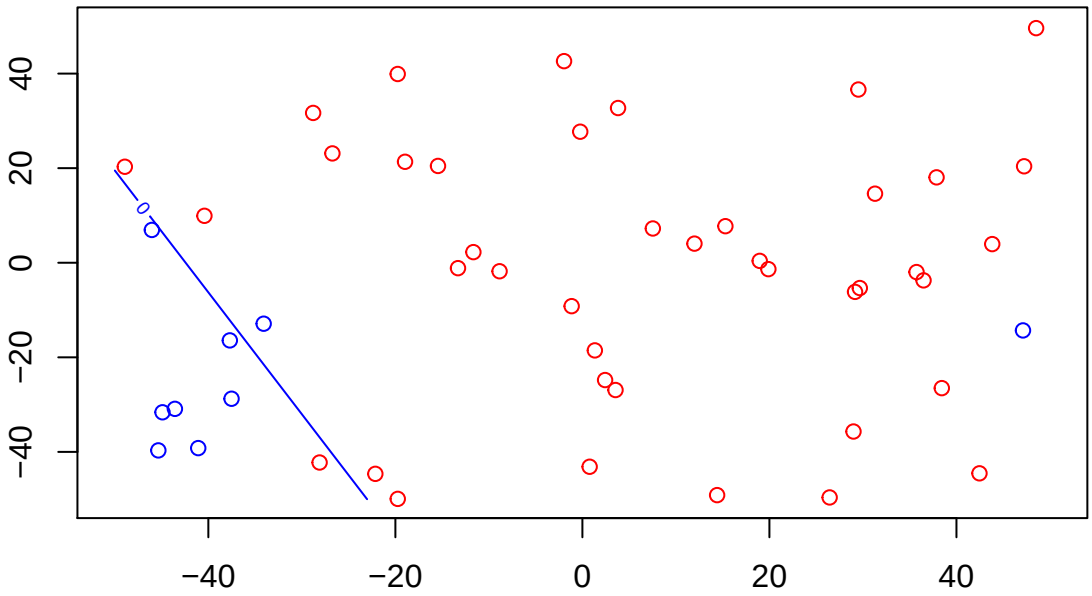
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_2[1]*x +sol_cuadratic_2[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_2+3))
```




```
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 47
```

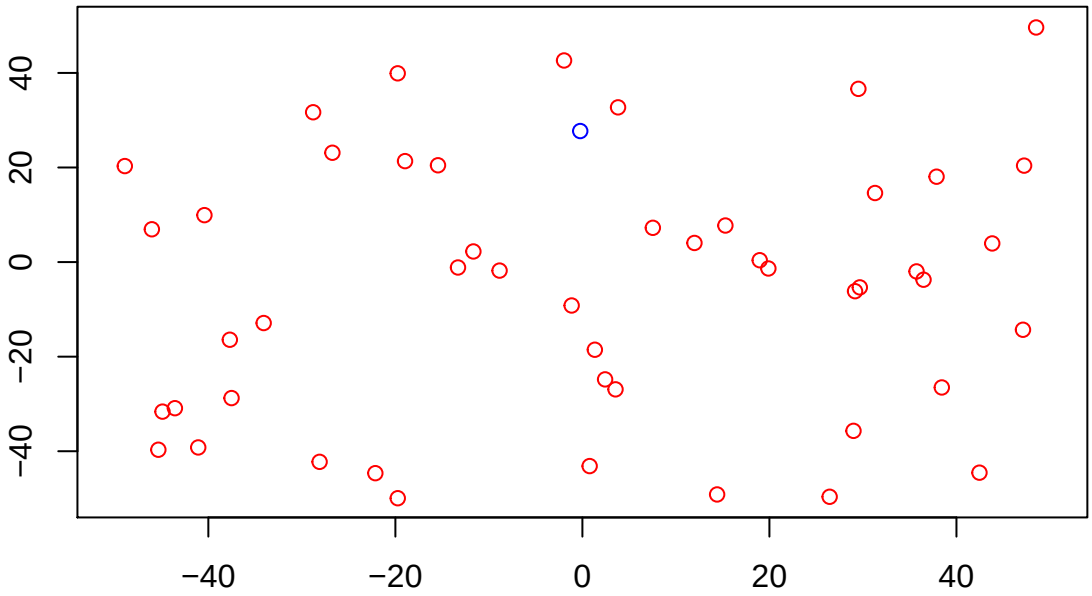
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_3[1]*x +sol_cuadratic_3[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_3+3))
```




```
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 1
```

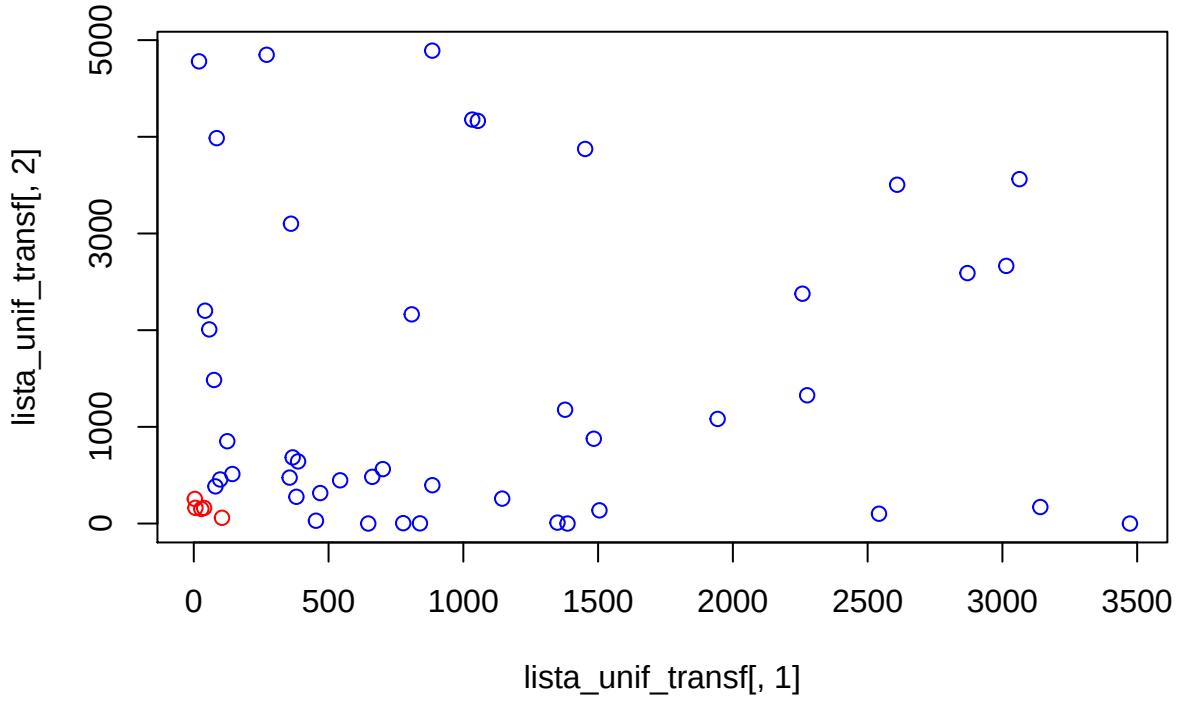
```
draw_function(c(-50,50), c(-50,50), function(x,y)
  y +sol_cuadratic_4[1]*x +sol_cuadratic_4[3], col = 4)
points(lista_unif[,1], lista_unif[,2], col = (etiqueta_4+3))
```



Se ve que sigue sin ser la mejor solución posible. Para separar estos datos lo que habría que hacer es buscar una transformación que los haga, al menos a excepción de los datos cuyas etiquetas hemos modificado aleatoriamente, separables. En este caso tendríamos que elevar cada coordenada (centrada en torno al centro o foco de la cuádrica) al cuadrado.

Por ejemplo, para el segundo caso, la transformación de los datos será $\Phi(x, y) = ((x-10)^2, (y-20)^2)$. De esta manera, los datos quedan de la forma:

```
lista_unif_transf <- matrix(apply(lista_unif,1,
   function(X)c((X[1]-10)^2,(X[2]-20)^2)),50,2,byrow=T)
plot(lista_unif_transf[,1], lista_unif_transf[,2], col = (etiqueta_1+3))
```

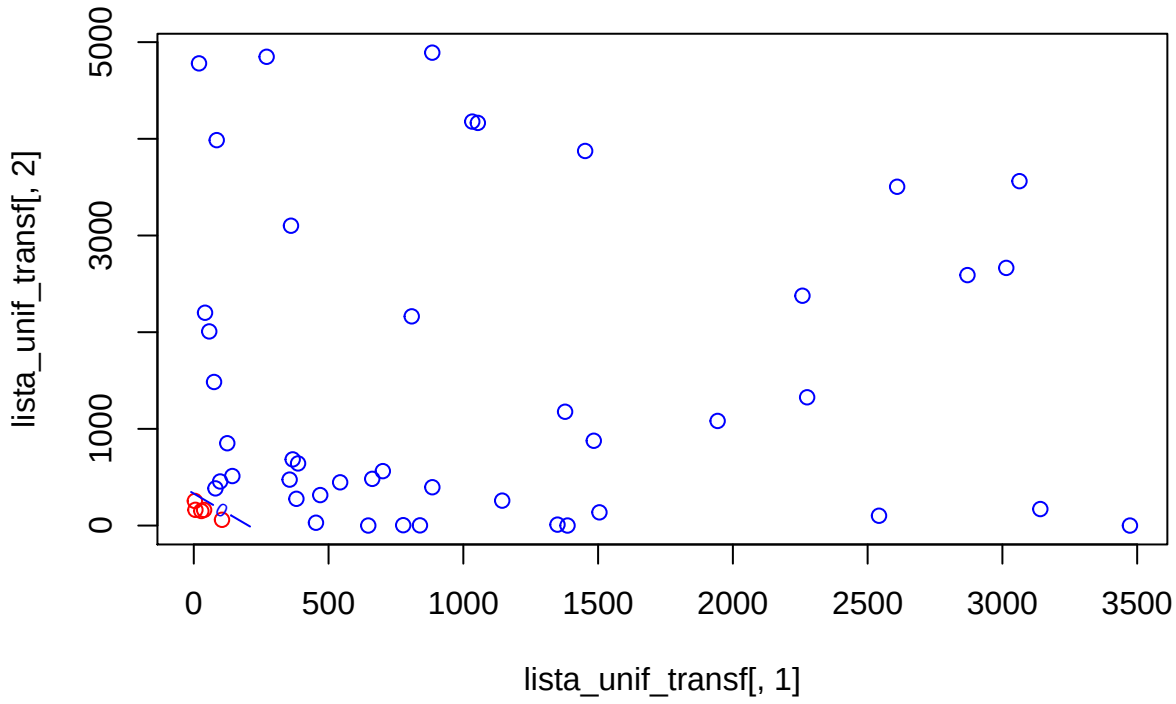


que es separable. Podemos aplicar ahora el algoritmo del Perceptron para separar los datos:

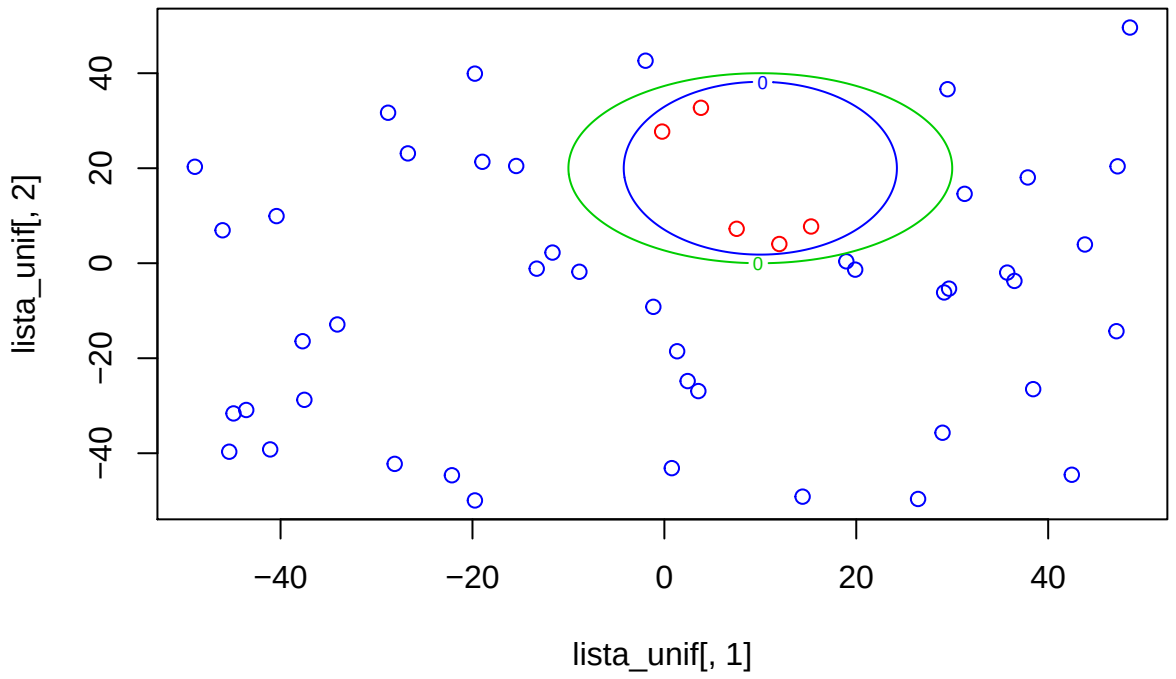
```
sol_transf <- sol_transf <- ajusta_PLA(lista_unif_transf, etiqueta_1, 4000, rep(0,3))$hiperplane
cat("Errores con la transformación:", count_errors(hiperplane_to_function(sol_transf),
                                lista_unif_transf, etiqueta_1 ))
```

```
## Errores con la transformación: 0
```

```
plot(lista_unif_transf[,1], lista_unif_transf[,2], col = (etiqueta_1+3))
draw_function(c(-10,1000), c(-10,1000), function(x,y)
  y +sol_transf[1]*x +sol_transf[3], col = 4,add=TRUE)
```




```
plot(lista_unif[,1], lista_unif[,2], col = (etiqueta_1+3))
draw_function(c(-50,50), c(-50,50), function(x,y)
    (y-20)^2 +sol_transf[1]*(x-10)^2 +sol_transf[3], col = 4,add=TRUE)
draw_function(c(-50,50), c(-50,50), f_1, col = 3,add=TRUE)
```



Vemos como, si sabemos o podemos intuir cuál es la transformación, podríamos aplicar el algoritmo PLA pese a que los datos no fuesen linealmente separables.

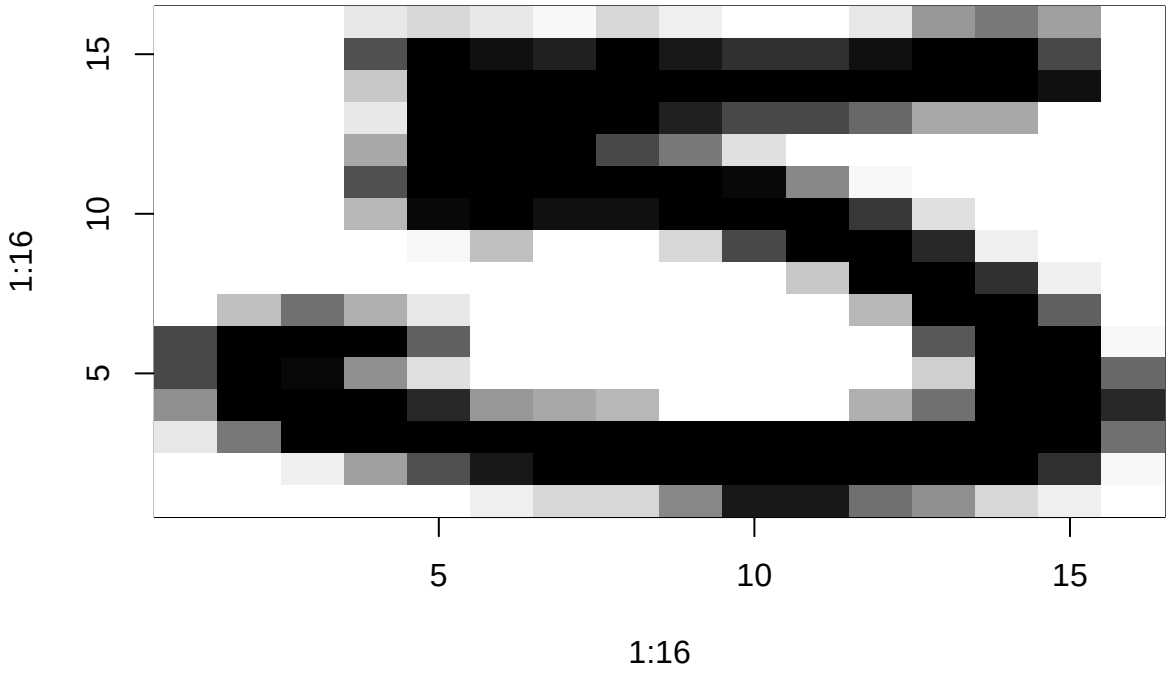
C - Regresión lineal

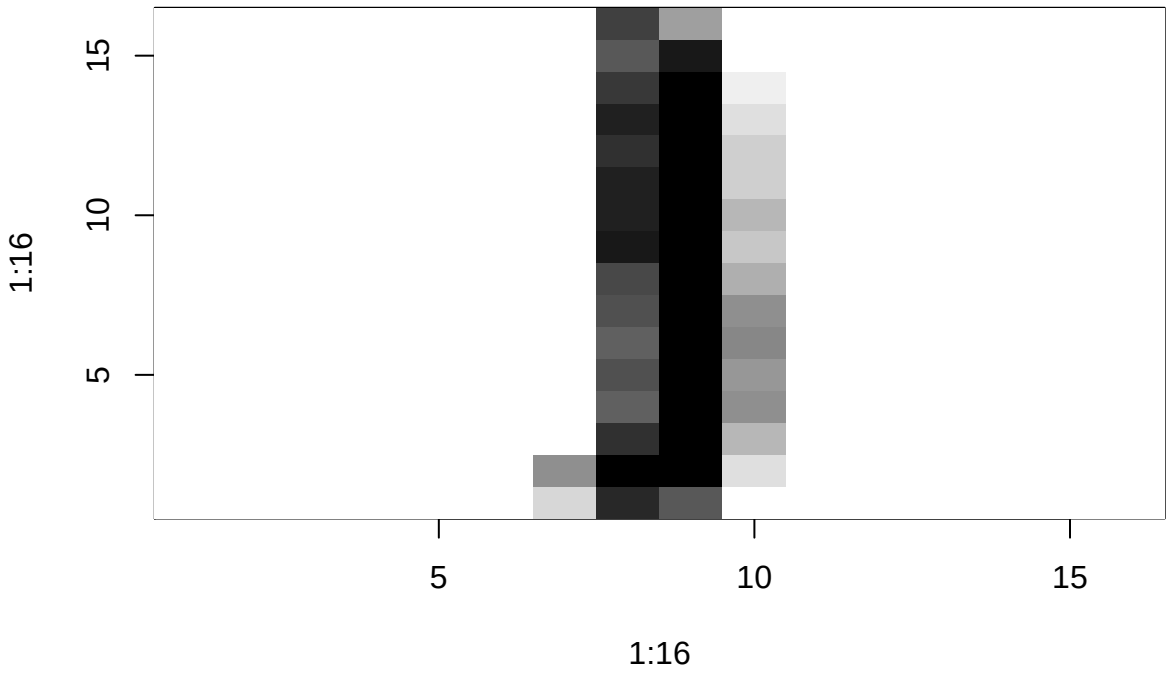
Leemos los datos del fichero `train.zip` y los colocamos en una matriz de 257 columnas en las que cada fila será un dato conformado por el número dibujado y 256 valores correspondientes a la intensidad en cada píxel. De estos datos seleccionamos únicamente aquellos en los que la primera columna contenga un 1 o un 5. Además, formamos un vector con los números dibujados, y un array tridimensional formado por matrices de tamaño 16×16 en las dos primeras dimensiones.

1 y 2. 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 <- array(t(datos[,2:257]), dim=c(16,16,nrow(datos)))
```

Para mostrar las imágenes hay que voltear las imágenes en el eje Y. Además le añadimos el parámetro `col` para ponerlas en blanco y negro y debido a que los valores en el fichero de datos son negativos, le damos la vuelta a la escala de grises:






```
for(i in 1:dim(pixels)[3]){
  image(1:16,1:16,pixels[,16:1,i],col = gray(32:0/32))
  Sys.sleep(0.05)
}
```

3. Para cada matriz de números calcularemos su valor medio y su grado de simetría vertical. Para calcular la simetría calculamos la suma del valor absoluto de las diferencias en cada píxel entre la imagen original y la imagen que obtenemos invertiendo el orden de las columnas. Finalmente le cambiamos el signo.

El vector de medias se obtiene directamente aplicando la función `mean` en cada elemento de la tercera dimensión. Para la simetría, creamos la función `simetria_vertical`, que calcula la simetría para cada matriz. Esta función, para cada fila, calcula la diferencia entre la fila y la fila invertida.

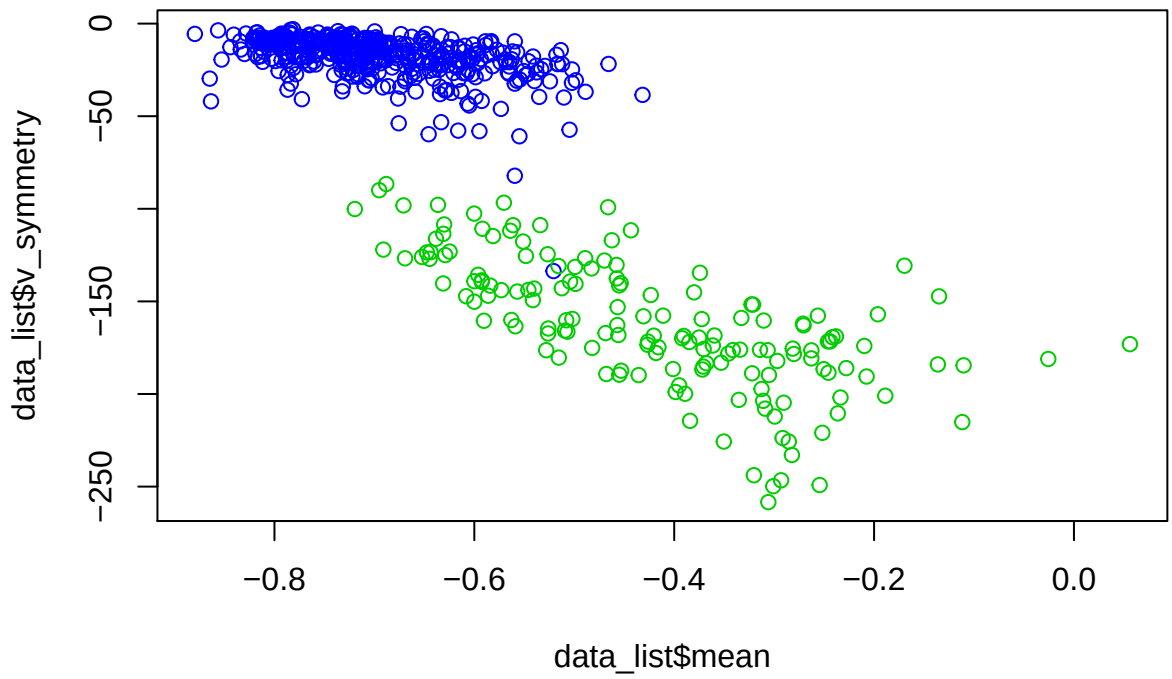
```
means <- apply(pixels, 3, mean)

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

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

4. Representación

Mostramos la gráfica con la simetría en el eje Y y la intensidad media en el eje X. Creamos también un vector con colores referentes al número que representan. Los 1s estarán representados por círculos azules y los 5s por círculos verdes.

```
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 )
```



5. 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.

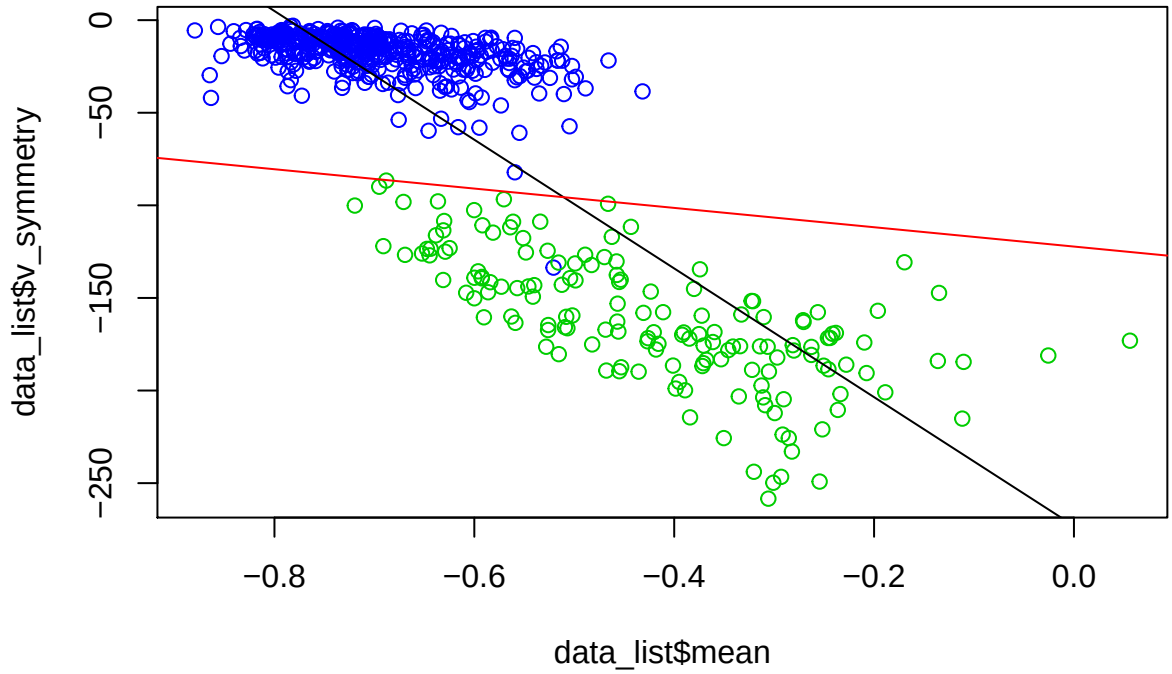
Creamos dos métodos para obtener la regresión lineal. La primera función calcula la pseudoinversa de una matriz. Para ello usamos la descomposición SVD e invertimos los valores en la diagonal que sean distintos de 0. Hay que mencionar que antes hacemos un redondeo a la quinta cifra decimal porque puede darse que haya un valor muy cercano a 0 (fruto de la precisión de la máquina) y que al invertirlo ese valor cobre demasiada relevancia.Para hacer la regresión lineal $Y \rightsquigarrow W^T X$, calculamos la matriz pseudoinversa de $X^T X$, $X^\star$ y entonces $W = X^\star X^T Y$. También podríamos tomar $X^\star$ como la pseudoinversa de X y $W = X^\star Y$

```
pseudoinv <- function(X){
  desc <- svd(X)
  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)
}

regress_lin <- function(datos, label){
  pseudo_inv <- pseudoinv(t(datos)%*%datos)
  return(pseudo_inv%*%t(datos)%*%label)
}
```

6. Ajustar un modelo de regresión lineal a los datos de (Intensidad promedio, Simetría) y pintar la solución junto con los datos. Valorar el resultado.

Se aprecia como la regresión lineal realizada en los datos sigue la nube de puntos , separándose de ellos cuando se aproxima a los 5s para quedar más cerca de los puntos lejanos a la nube de puntos. Si además lo que queremos hacer es separar los datos, tenemos que incluir las etiquetas para que la regresión lineal separe los datos, con buenos resultados teniendo en cuenta que los datos no son separables.

```
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])

label <- numeric(length(number))
label[which(number==1)] <- -1
label[which(number==5)] <- 1
coefs2 <- regress_lin(cbind(means,sim_vertical,rep(1,length(means))), label)
abline(-coefs2[3]/coefs2[2],-coefs2[1]/coefs2[2],col=2)
```



7. 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

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

Debido a que estos dos apartados consecutivos tienen la misma intención y sólo varía la muestra y sobre cuál se hace la regresión, creo que tiene sentido hacerlos conjuntamente. En primer lugar se simula una recta

de la manera que lo hemos venido haciendo, con `simula_recta`. Generamos una muestra aleatoria de 100 datos y la etiquetamos según el signo con respecto a la recta. Entonces realizamos la regresión lineal y vemos cuántos fallos hay con respecto a las etiquetas originales. Para los datos de fuera de la muestra, generamos unos datos aleatorios, vemos la etiquieta que asignaría la regresión lineal y la recta original. La diferencia aquí es que la regresión no se ha realizado con respecto a estos datos sino con respecto a los originales. El error obtenido en la muestra es de un 4.072%, mientras que fuera de la muestra es de 4.945%. La diferencia se explica porque la regresión se ha realizado con los datos que consideramos dentro de la muestra.

```
N <- 100
num_repeticiones <- 1000
E_in_vector <- numeric(num_repeticiones)
E_out_vector <- numeric(num_repeticiones)

for( i in 1:num_repeticiones){
  f <- simula_recta(c(-10,10))

  muestra_in <- simula_unif(N, 2, c(-10, 10))
  etiqueta_in <- apply(muestra_in, 1, function(X) sign(f(X[1],X[2])))

  coefs_rl <- regress_lin(cbind(muestra_in[,1],muestra_in[,2], rep(1,N)),
                          matrix(etiqueta_in,N,1))
  g <- function(x,y){coefs_rl[2]*y + coefs_rl[1]*x + coefs_rl[3]}

  rl_etiqueta_in <- apply(muestra_in, 1, function(X) sign(g(X[1],X[2])))
  E_in <-sum(etiqueta_in != rl_etiqueta_in)/N*100
  E_in_vector[i] <- E_in

  muestra_out <- simula_unif(N, 2, c(-10, 10))
  etiqueta_out <- apply(muestra_out, 1, function(X) sign(f(X[1],X[2])))
  rl_etiqueta_out <- apply(muestra_out, 1, function(X) sign(g(X[1],X[2])))
  E_out <- sum(etiqueta_out != rl_etiqueta_out)/N*100
  E_out_vector[i] <- E_out
}

cat( "Ein. Media = ", mean(E_in_vector), "\n Max = ", max(E_in_vector),
       "\n Min = ", min(E_in_vector), "\n Desv. Típica = ", sd(E_in_vector))
```

```
## Ein. Media =  4.072
##  Max =  14
##  Min =  0
##  Desv. Típica =  2.970124
```

```
cat( "Eout. Media = ", mean(E_out_vector), "\n Max = ", max(E_out_vector),
       "\n Min = ", min(E_out_vector), "\n Desv. Típica = ", sd(E_out_vector))
```

```
## Eout. Media =  4.945
##  Max =  22
##  Min =  0
##  Desv. Típica =  3.793991
```

c) Ahora fijamos N = 10, ajustamos regresión lineal y usamos el vector de pesos encontrado como un vector inicial de pesos para PLA. Ejecutar PLA hasta que converja a un vector

de pesos final que separe completamente la muestra de entrenamiento. Anote el número de iteraciones y repita el experimento 1.000 veces ¿Cuál es valor promedio de iteraciones que tarda PLA en converger? (En cada iteración de PLA elija un punto aleatorio del conjunto de mal clasificados). Valore los resultados.

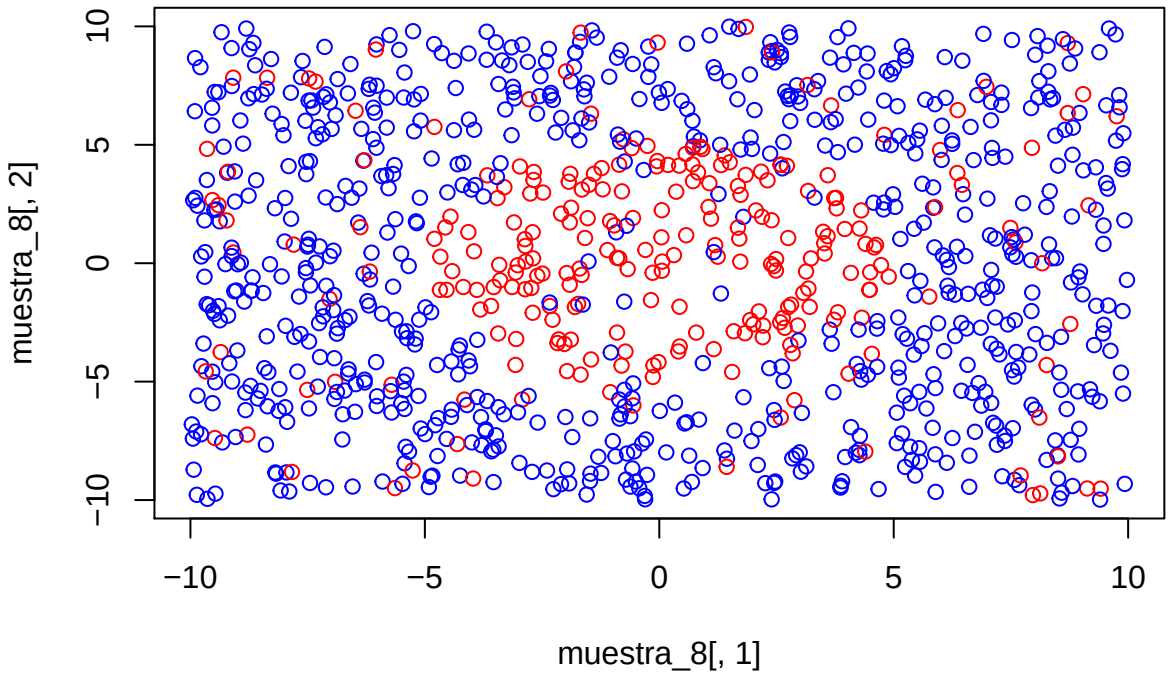
Ahora lo que hacemos es ejecutar el PLA una vez realizada la regresión lineal. De nuevo se simula una recta y se realiza una regresión lineal sobre los datos. Entonces pasamos el vector solución dado por la regresión lineal y contamos el número de iteraciones hasta que converje, que sabremos que pasará debido a que los datos son separables. Vemos que se tienen de media 9.599 iteraciones, sin embargo cabría esperar tener menos ya que los datos estarían clasificados.

```
N <- 10
num_repeticiones <- 1000
iterations_vector <- numeric(num_repeticiones)

for( i in 1:num_repeticiones){
  f <- simula_recta(c(-10,10))

  muestra_in <- simula_unif(N, 2, c(-10, 10))
  etiqueta_in <- apply(muestra_in, 1, function(X) sign(f(X[1],X[2])))
  coefs_rl <- regress_lin(cbind(muestra_in[,1],muestra_in[,2], rep(1,N)),
                          matrix(etiqueta_in,N,1))

  iterations[i] <- ajusta_PLA(muestra_in,etiqueta_in, 10000,coefs_rl)$iterations
}

cat( "Iterations. Media = ", mean(iterations), "\n Max = ", max(iterations),
       "\n Min = ", min(iterations), "\n Desv. Típica = ", sd(iterations))
```

```
## Iterations. Media =  9.599
##  Max =  1291
##  Min =  1
##  Desv. Típica =  50.4776
```

8. En este ejercicio exploramos el uso de transformaciones no lineales. Consideremos la función objetivo $f(x_1, x_2) = sign(x_1^2 + x_2^2 - 25)$. Generar una muestra de entrenamiento de $N = 1000$ puntos a partir de $\mathcal{X} = [10, 10] \times [10, 10]$ muestreando cada punto $x \in \mathcal{X}$ uniformemente. Generar las salidas usando el signo de la función en los puntos muestreados. Generar ruido sobre las etiquetas cambiando el signo de las salidas a un 10 de puntos del conjunto aleatorio generado.

```
N <- 1000
f <- function(x,y){
  x^2+y^2-25
}

muestra_8 <- simula_unif(N, 2, c(-10, 10))
etiquetas_8 <- modify_rnd_bool_subvector(apply(muestra_8, 1, function(X) sign(f(X[1],X[2]))))
plot(muestra_8[,1],muestra_8[,2],col=etiquetas_8+3)
```



a) Ajustar regresion lineal, para estimar los pesos w. Ejecutar el experimento 1.000 veces y calcular el valor promedio del error de entrenamiento E_{in} . Valorar el resultado.

```
for( i in 1:num_repeticiones){
  muestra_8 <- simula_unif(N, 2, c(-10, 10))
  etiquetas_8 <- modify_rnd_bool_subvector(apply(muestra_8, 1, function(X) sign(f(X[1],X[2]))))

  coefs_rl <- regress_lin(cbind(muestra_8[,1],muestra_8[,2], rep(1,N)),
                          matrix(etiquetas_8,N,1))
  g <- function(x,y){coefs_rl[2,1]*y + coefs_rl[1,1]*x + coefs_rl[3,1]}

  rl_etiqueta_8 <- apply(muestra_8, 1, function(X) sign(g(X[1],X[2])))
  E_in <-sum(etiquetas_8 != rl_etiqueta_8)/N*100
  E_in_vector[i] <- E_in
}
cat( "Ein. Media = ", mean(E_in_vector), "\n Max = ", max(E_in_vector),
       "\n Min = ", min(E_in_vector), "\n Desv. Típica = ", sd(E_in_vector))
```

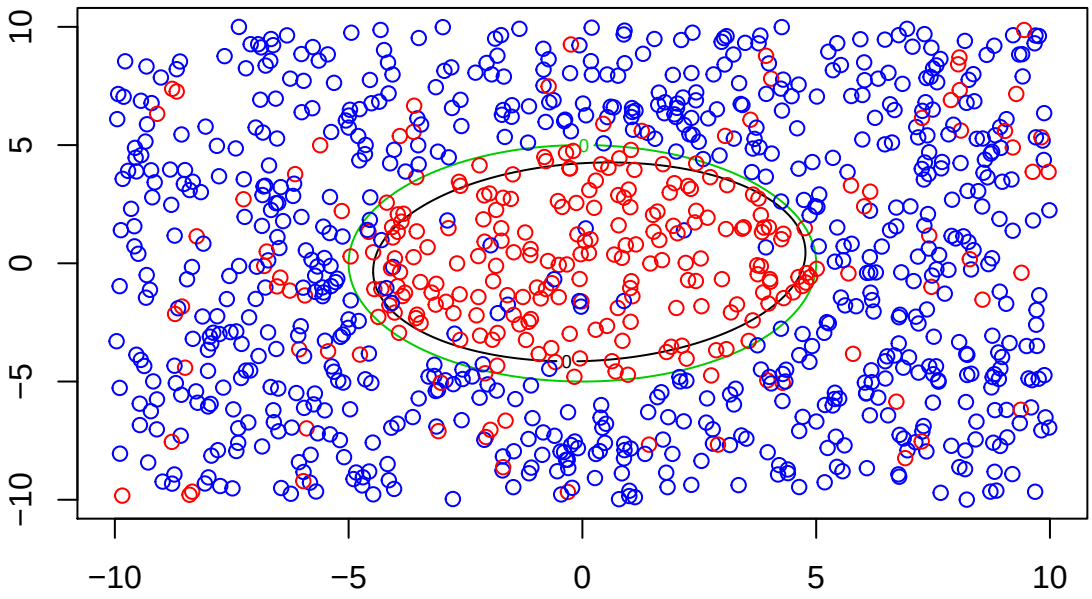
```
## Ein. Media =  25.6571
##  Max =  29.3
##  Min =  22.4
##  Desv. Típica =  1.011253
```

Como vemos el error está en torno al 25%, lo que significa que no realiza una clasificación aceptable. Esto se debe a que los datos no son separables linealmente y estamos intentando hacerlo mediante una función lineal, lo que nos da este error. Veremos en el siguiente apartado que el resultado es mucho mejor.

b) y c) Ahora,consideremos $N = 1000$ datos de entrenamiento y el siguiente vector de variables: $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ajustar de nuevo regresión lineal y calcular el nuevo vector de pesos $\hat{w}$. Mostrar el resultado.

De nuevo tiene más sentido hacer estos dos apartados simultáneamente. Vemos que el ajuste es mucho mejor, pues dentro de la muestra (de cuyos valores hemos cambiado un 10%) el error es de un 15%. En cambio, para valores de fuera de la muestra con la que se ha realizado la regresión lineal, el error se reduce a un 7%. Esta bajada del porcentaje de error se debe a que no se han modificado las etiquetas como sí se ha hecho en los datos de la muestra. Si pintamos por ejemplo en la última iteración los datos, la función original y la estimada por la regresión lineal, se comprueba que la diferencia podría ser aceptable teniendo en cuenta que la regresión se ha realizado con datos con ruido.

```
for( i in 1:num_repeticiones){
  muestra_8 <- simula_unif(N, 2, c(-10, 10))
  etiquetas_8 <- modify_rnd_bool_subvector(apply(muestra_8, 1, function(X) sign(f(X[1],X[2]))))

  coefs_rl <- regress_lin(cbind(muestra_8[,1],muestra_8[,2],muestra_8[,1]*muestra_8[,2],
                                muestra_8[,1]**2,muestra_8[,2]**2,rep(1,N)),
                          matrix(etiquetas_8,N,1))
  g <- function(x,y){coefs_rl[1]*x + coefs_rl[2]*y +coefs_rl[3]*x*y +
                       coefs_rl[4]*x^2 + coefs_rl[5]*y^2 + coefs_rl[6]}

  rl_etiqueta_in <- apply(muestra_8, 1, function(X) sign(g(X[1],X[2])))
  E_in <-sum(etiquetas_8 != rl_etiqueta_in)/N*100
  E_in_vector[i] <- E_in

  muestra_8_out <- simula_unif(N, 2, c(-10, 10))
  etiqueta_8_out <- apply(muestra_8_out, 1, function(X) sign(f(X[1],X[2])))
  rl_etiqueta_8_out <- apply(muestra_8_out, 1, function(X) sign(g(X[1],X[2])))
  E_out <- sum(etiqueta_8_out != rl_etiqueta_8_out)/N*100
  E_out_vector[i] <- E_out
}
draw_function(c(-10,10), c(-10,10), f, col = 3)
draw_function(c(-10,10), c(-10,10), g, col = 1, add = TRUE)
points(muestra_8[,1],muestra_8[,2],col=etiquetas_8+3)
```




```
cat( "Ein. Media = ", mean(E_in_vector), "\n Max = ", max(E_in_vector),
       "\n Min = ", min(E_in_vector), "\n Desv. Típica = ", sd(E_in_vector))
```

```
## Ein. Media =  15.4515
##  Max =  23.1
##  Min =  10.4
##  Desv. Típica =  2.272625
```

```
cat( "Eout. Media = ", mean(E_out_vector), "\n Max = ", max(E_out_vector),
       "\n Min = ", min(E_out_vector), "\n Desv. Típica = ", sd(E_out_vector))
```

```
## Eout. Media =  6.9718
##  Max =  17.4
##  Min =  0.3
##  Desv. Típica =  3.1561
```