Aprendizaje Automático (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 2: Programación

Laura Tirado López

30 de abril de 2016

Parte I.
MODELOS LINEALES

1. Gradiente Descendente. Implementar el algoritmo de gradiente descendiente.

1.1. Considerar la función no lineal de error $E(u,v) = (ue^v 2ve^{-u})^2$ descendente y minimizar esta función de error, comenzando desde el punto $(u,v) = (1,1)$ y usando una tasa de aprendizaje $\mu = 0,1$.

1.1.1. Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u,v)$

Calculamos la derivada de E respecto de u y respecto de v.

Derivada de E respecto de u:

Tratamos v como constante y aplicamos la regla de la cadena siendo $w = (ue^v - 2ve^{-u})$.

$$\frac{\partial}{\partial u}((ue^v - 2ve^{-u})^2) = 2e^{-2u} = \frac{\partial}{\partial w}(w^2)\frac{\partial}{\partial u}(ue^v - 2ve^{-u})$$

Siendo:

$$\frac{\partial}{\partial w}(w^2) = 2w$$

$$\frac{\partial}{\partial u}(ue^v - 2ve^{-u}) = 2ve^{-u} + e^v$$

Por tanto:

$$\frac{\partial}{\partial w}(w^2)\frac{\partial}{\partial u}(ue^v - 2ve^{-u}) = 2w(2ve^{-u} + e^v)$$

Sustituimos $w = (ue^v - 2ve^{-u})$:

$$2w(2ve^{-u} + e^v) = 2(ue^v - 2ve^{-u})(2ve^{-u} + e^v)$$

Simplificamos:

$$\frac{\partial}{\partial u}((ue^v - 2ve^{-u})^2) = 2e^{-2u}(e^{u+v}u - 2v)(2v + e^{u+v})$$

Derivada de E respecto de v:

Tratamos u como constante y aplicamos la regla de la cadena siendo $w = (ue^v - 2ve^{-u})$.

$$\frac{\partial}{\partial v}((ue^v - 2ve^{-u})^2) = 2e^{-2u} = \frac{\partial}{\partial w}(w^2)\frac{\partial}{\partial v}(ue^v - 2ve^{-u})$$

Siendo:

$$\frac{\partial}{\partial w}(w^2) = 2w$$

$$\frac{\partial}{\partial v}(ue^v - 2ve^{-u}) = ue^v - 2e^{-u}$$

Por tanto:

$$\frac{\partial}{\partial w}(w^2)\frac{\partial}{\partial v}(ue^v - 2ve^{-u}) = 2w(ue^v - 2e^{-u})$$

Sustituimos $w = (ue^v - 2ve^{-u})$:

$$2w(ue^v - 2e^{-u}) = 2(ue^v - 2ve^{-u})(ue^v - 2e^{-u})$$

Simplificamos:

$$\frac{\partial}{\partial v}((ue^v - 2ve^{-u})^2) = 2e^{-2u}(e^{u+v}u - 2)(e^{u+v}u - 2v)$$

La expresión del gradiente sería:

$$\bigtriangledown f(x) = (\frac{\partial}{\partial u}, \frac{\partial}{\partial v}) = (2e^{-2u}(e^{u+v}u - 2v)(2v + e^{u+v}), 2e^{-2u}(e^{u+v}u - 2)(e^{u+v}u - 2v))$$

1.1.2. ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u, v)$ inferior a 10^{14}?. (Usar flotantes de 64 bits)

El algoritmo tarda 10 iteraciones en encontrar por primera vez un valor de $E(u, v)$ inferior a 10^{14}.

1.1.3. ¿Qué valores de (u, v) obtuvo en el apartado anterior cuando alcanzo el error de 10^{-14}?

Los valores de (u, v) son $(0, 04473629, 0, 02395871)$.

1.2. Considerar ahora la función $f(x, y) = x^2 + 2y^2 + 2sin(2\pi x)sin(2\pi y)$

1.2.1. Usar gradiente descendente para minimizar esta función. Usar como valores iniciales $x_0 = 1$, $y_0 = 1$, la tasa de aprendizaje $\mu = 0, 01$ y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando $\mu = 0, 1$, comentar las diferencias.

Para mostrar un gráfico, modificamos la función *gradiente_descendente* añadiendo las lineas necesarias para pintar la función y los puntos que se van generando. El código sería el siguiente:

```
gradiente_descendente <-function(w,f,df,mu,it_max){

  num_it <- 0
  aux <- c(0,0)
  primera_vez <- TRUE
  x <- seq(from=-5,to=5,by=0.25)
  y <- seq(from=-5,to=5,by=0.25)
  m <- matrix(w,nrow=1)

  #Dibujamos la funci n y el punto inicial
  contour(x,y,outer(x,y,Vectorize(function(x,y){f(x,y)})),xlab="
     Eje X",ylab="Eje Y",col='black')
  w1 <- df(w[1],w[2])
  points(w[1],w[2],pch=21,bg='blue')

  for (i in 1:it_max){

    # Actualizamos w seg n el valor de la derivada
    w <- w - df(w[1],w[2])*mu
    m <- rbind(m,w)

    if (f(w[1],w[2]) < 10**(-14) & primera_vez) {
      num_it <- i
      aux <- w
      primera_vez <- FALSE
    }

    # Dibujamos el nuevo punto
    w1 <- df(w[1],w[2])
```

```
29      points(w[1],w[2],col='red')
30
31    }
32
33    # Dibujamos el punto final
34    w1 <- df(w[1],w[2])
35    points(w[1],w[2],pch=21,bg='green')
36
37    # Dibujamos el camino
38    lines(m[,1],m[,2],col='red')
39
40    return(list(w,num_it,aux))
41  }
```

El gráfico para una tasa de aprendizaje $\mu = 0,01$ es el siguiente:

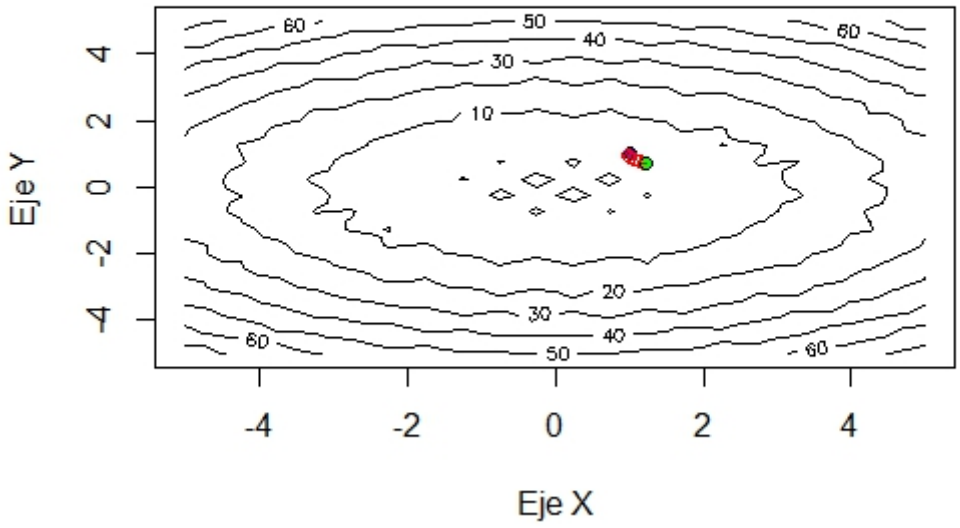

Figura 1.1: Gráfico de gradiente descendente para $\mu = 0,01$

El punto azul indica el punto de inicio y el verde el punto final. Los puntos rojos son los distintos puntos que se han ido generando conforme descendía el valor de la función.

El gráfico para una tasa de aprendizaje $\mu = 0,1$ es el siguiente:

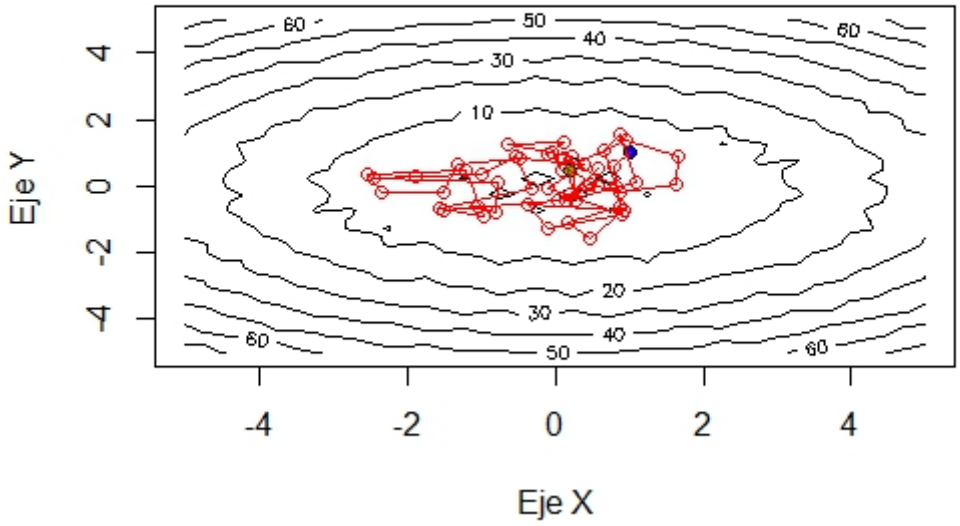

Figura 1.2: Gráfico de gradiente descendente para $\mu = 0,1$

La principal diferencia que podemos observar es que para una tasa de aprendizaje $\mu = 0,1$ el algoritmo tarda más en converger que con la tasa $\mu = 0,01$. Aunque con la tasa $\mu = 0,1$, se encuentra un mínimo más pequeño.

1.2.2. Obtener el valor mínimo y los valores de las variables que lo alcanzan cuando el punto de inicio se fija: $(0,1,0,1)$, $(1,1)$, $(0,5,0,5)$,$(1,1)$. Generar una tabla con los valores obtenidos ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

Punto de inicio	$f(x,y)$	**Coordenadas**
$(0,1,0,1)$	1,957033	$[0{,}1003917, -1{,}0157510]$
$(1,1)$	$-1{,}699786$	$[0{,}1973570, -0{,}2667825]$
$(0,5,0,5)$	2,070314	$[0{,}1729822, -0{,}9943370]$
$(1,1)$	$-1{,}699786$	$[-0{,}1973570, 0{,}2667825]$

Tabla 1.1: Tabla de valores mínimos

A partir de los datos de la tabla, podemos deducir que el punto de inicio influye considerablemente en encontrar el mínimo global de la función.

2. Coordenada descendente. En este ejercicio comparamos la eficiencia de la técnica de optimización de "coordenada descendente" usando la misma función del ejercicio 1.1a. En cada iteración, tenemos dos pasos a lo largo de dos coordenadas. En el $Paso-1$ nos movemos a lo largo de la coordenada u para reducir el error (suponer que se verifica una aproximación de primer orden como en gradiente descendente), y el $Paso-2$ es para reevaluar y movernos a lo largo de la coordenada v para reducir el error (hacer la misma hipótesis que en el $paso-1$). Usar una tasa de aprendizaje $\mu = 0,1$.

2.1. ¿Qué valor de la función $E(u,v)$ se obtiene después de 15 iteraciones completas (i.e. 30 pasos)?

El código del algoritmo sería el siguiente:

```
coordenada_descendente <-function(w,f,df,mu,it_max){

  x <- seq(from=-5,to=5,by=0.25)
  y <- seq(from=-5,to=5,by=0.25)
  m <- matrix(w,nrow=1)

  #Dibujamos la funci n y el punto inicial
  contour(x,y,outer(x,y,Vectorize(function(x,y){f(x,y)})),xlab="
     Eje X",ylab="Eje Y",col='black')
  w1 <- df(w[1],w[2])
  points(w[1],w[2],pch=21,bg='blue')

  for (i in 1:it_max){

    #Calculamos las derivadas parciales
    du <- (df(w[1],w[2]))[1]
    dv <- (df(w[1],w[2]))[2]

    # Actualizamos seg n el valor de la derivada
    if(i%%2 == 0)
      w[1] <- w[1] - du*mu
    else
      w[2] <- w[2] - dv*mu

    m <- rbind(m,w)

```

```
26      # Dibujamos el nuevo punto
27      w1 <- df(w[1],w[2])
28      points(w[1],w[2],col='red')
29    }
30
31    # Dibujamos el punto final
32    w1 <- df(w[1],w[2])
33    points(w[1],w[2],pch=21,bg='green')
34
35    # Dibujamos el camino
36    lines(m[,1],m[,2],col='red')
37
38    return(list(f(w[1],w[2]),w))
39  }
```

Después de 15 iteraciones completas el valor que obtenemos es $1,523901 \cdot 10^{-15}$

2.2. Establezca una comparación entre esta técnica y la ténica del gradiente descendente.

Para establecer una comparación entre ambas técnicas, he calculado el valor de la función despues de 15 iteraciones para ambas y además he sacado las gráficas del descenso.

El valor de la función para el algoritmo de coordenada descendente es $1,523901 \cdot 10^{-15}$ y para gradiente descendente $6{,}026781 \cdot 10^{-27}$.

La gráficas de descenso son las siguientes:

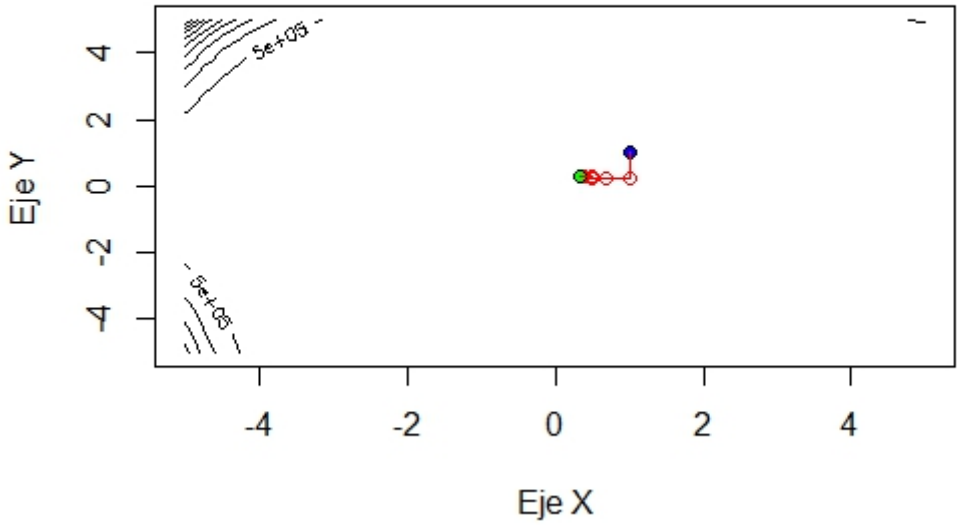

Figura 2.1: Gráfica de descenso para coordenada descendente

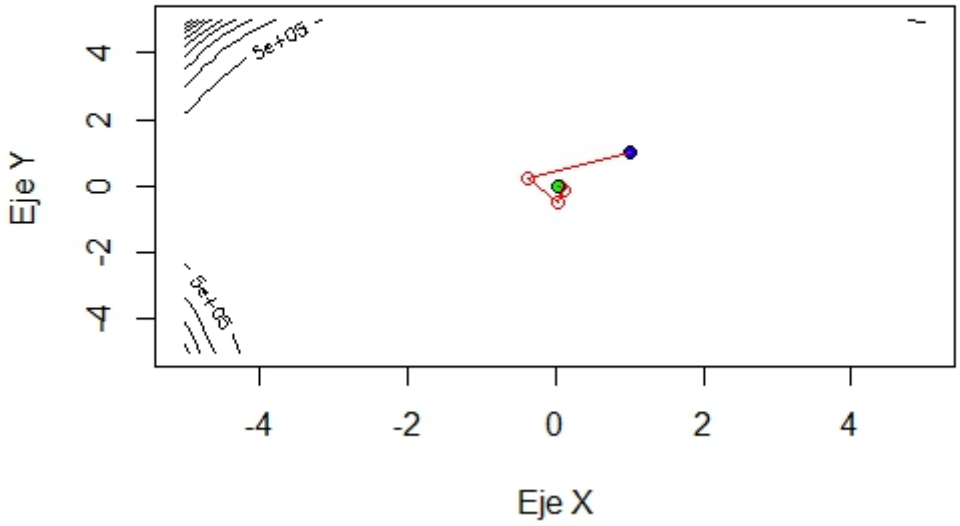

Figura 2.2: Gráfica de descenso para gradiente descendente

Si nos fijamos en los valores de la función podemos ver como el gradiente descendente alcanza un mínimo más pequeño que la coordenada descendente. Además, en las gráfica observamos que el gradiente descendente parece converger más rápidamente que la coordenada descendente.

Según estos datos, el algoritmo de gradiente descendente parece ser significativamente mejor que el de coordenada descendente.

3. Método de Newton.Implementar el algoritmo de minimización de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio.1b. Desarrolle los mismos experimentos usando los mismos puntos de inicio.

- Generar un gráfico de como desciende el valor de la función con las iteraciones.
- Extraer conclusiones sobre las conductas de los algoritmos comparando la curva de decrecimiento de la función calculada en el apartado anterior y la correspondiente obtenida con gradiente descendente.

El código del algoritmo es el siguiente:

```
newton <-function(w,f,df,ddf,mu,it_max){

  x <- seq(from=-5,to=5,by=0.25)
  y <- seq(from=-5,to=5,by=0.25)
  m <- matrix(w,nrow=1)

  #Dibujamos la funci n y el punto inicial
  contour(x,y,outer(x,y,Vectorize(function(x,y){f(x,y)})),xlab="
     Eje X",ylab="Eje Y",col='black')
  w1 <- df(w[1],w[2])
  points(w[1],w[2],pch=21,bg='blue')

  for (i in 1:it_max){
    # Calculamos la funci n Hessiana
    H <- matrix(ddf(w[1],w[2]),nrow=2)

    # Actualizamos w
    w <- solve(H)*(w-df(w[1],w[2])*mu)
    m <- rbind(m,w)

    # Dibujamos el nuevo punto
    w1 <- df(w[1],w[2])
    points(w[1],w[2],col='red')
  }

  # Dibujamos el punto final
  w1 <- df(w[1],w[2])
  points(w[1],w[2],pch=21,bg='green')

  # Dibujamos el camino
  lines(m[,1],m[,2],col='red')

  return(c(f(w[1],w[2]),w))
}
```

La función *ddf* nos calcula los valores de las derivadas de segundo orden. El código es el siguiente:

```
ddf <- function(x,y){

  dxx <- 2-8*pi**2*sin(2*pi*x)*sin(2*pi*y)
  dxy <- 8*pi**2*cos(2*pi*x)*cos(2*pi*y)
  dyx <- 8*pi**2*cos(2*pi*x)*cos(2*pi*y)
  dyy <- 4-8*pi**2*sin(2*pi*x)*sin(2*pi*y)

  return (c(dxx,dxy,dyx,dyy))
}
```

A continuación se muestran los gráficos de descenso para los cuatro ejemplos con los dos algoritmos:

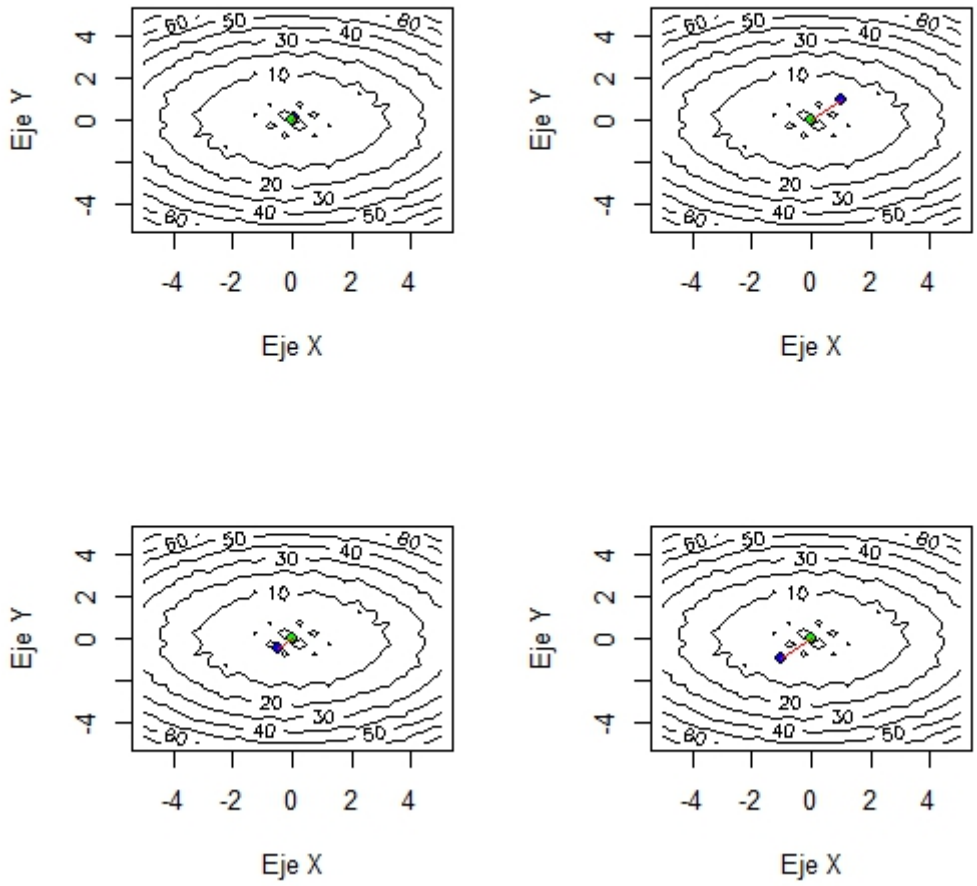

Figura 3.1: Gráficas de descenso para el algoritmo de Newton

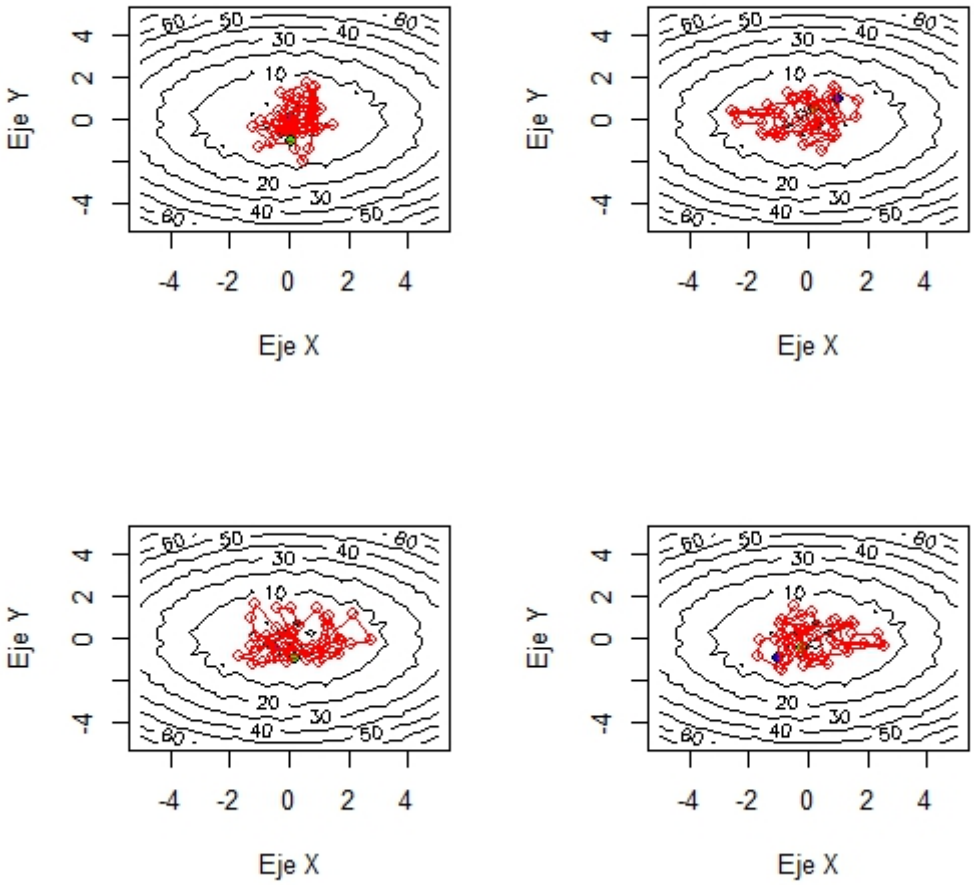

Figura 3.2: Gráficas de descenso para el algoritmo de gradiente descendente

Al comparar las curvas de descenso, podemos ver como claramente el algoritmo de Newton converge más rápidamente que el algoritmo de gradiente descendente. Además, podemos ver como en todos los ejemplos alcanza el mismo mínimo, al contrario que el algoritmo de gradiente descendente.

La conducta del algoritmo de Newton no depende el punto de inicio, mientras que la del algoritmo de gradiente descendiente parece ser dependiente del punto de inicio a la hora de encontrar el mínimo global.

4. Regresión Logística: En este ejercicio crearemos nuestra propia función objetivo f (probabilidad en este caso) y nuestro conjunto de datos D para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores $0/1$ y por tanto que y es una función determinista de x.

Consideremos $d = 2$ para que los datos sean visualizables, y sea $X = [1, 1][1, 1]$ con probabilidad uniforme de elegir cada $x \in X$. Elegir una línea en el plano como la frontera entre $f(x) = 1$ (donde y toma valores $+1$) y $f(x) = 0$ (donde y toma valores 1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos.
Seleccionar $N = 100$ puntos aleatorios x_n de X y evaluar las respuestas de todos ellos y_n respecto de la frontera elegida.

4.1. Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD) bajo las siguientes condiciones:

- Inicializar el vector de pesos con valores 0.
- Parar el algoritmo cuando $|w(t1)w(t)|| < 0,01$, donde $w(t)$ denota el vector de pesos al final de la época t. Una época es un pase completo a través de los N datos.
- Aplicar una permutación aleatoria de $1, 2, ..., N$ a los datos antes de usarlos en cada época del algoritmo.
- Usar una tasa de aprendizaje de $\mu = 0,01$.

```
regresion_logistica <-function(w,x,y,N,mu){

  w_ant = c(1,1,1)
  x <- cbind(x,1)

  while(sqrt(sum((w_ant-w)**2)) >= 0.01){

    indices <- sample(1:length(y),N)
    x_muestra <- x[indices,]
    y_muestra <- y[indices]
    w_ant <- w

    # Calculamos el valor del gradiente
```

```
14      gradient <- (-y_muestra*x_muestra)/c(1+exp(y_muestra*x_muestra
            %*%w))
15
16      # Actualizamos w seg?n el valor de la derivada
17      w <- w-mu*1/N*colSums(gradient)
18    }
19
20    return(w)
21  }
```

Tras probarlo con una muestra aleatoria y el umbral ajustado a $|w(t1)w(t)|| < 0,001$, el resultado se muestra en la siguiente figura:

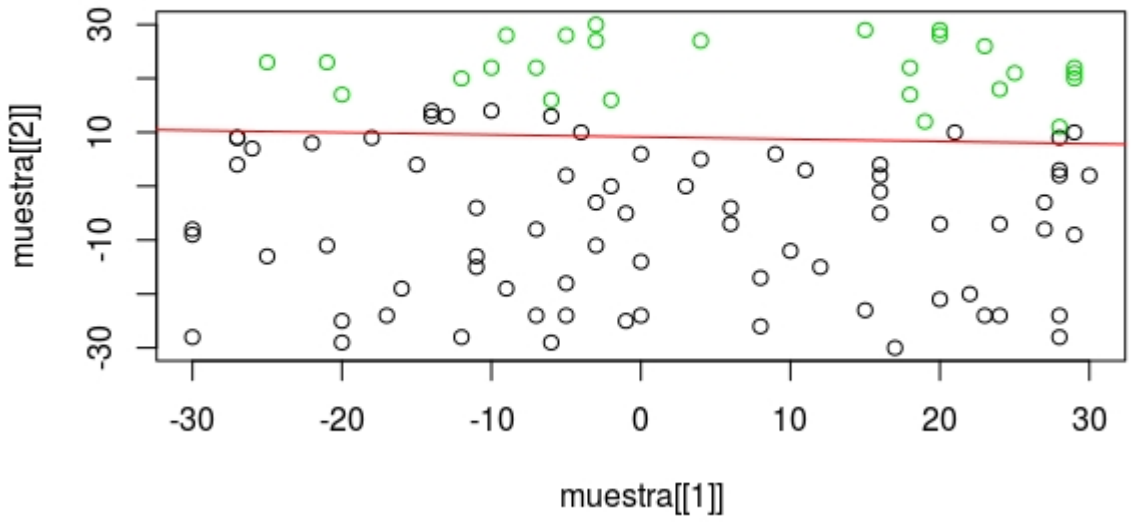

Figura 4.1: Gráficas de regresión logística

4.2. Usar la muestra de datos etiquetada para encontrar g y estimar E_{out} usando para ello un número suficientemente grande de nuevas muestras.

Con una muestra de $N = 1000$, el resultado es el siguiente:

- La función g es $g(x) = 3,11x + 12,3402$.
- El error fuera de la muestra es $8,9\,\%$.

En total cometió 83 fallos sobre una muestra de 1000 datos.

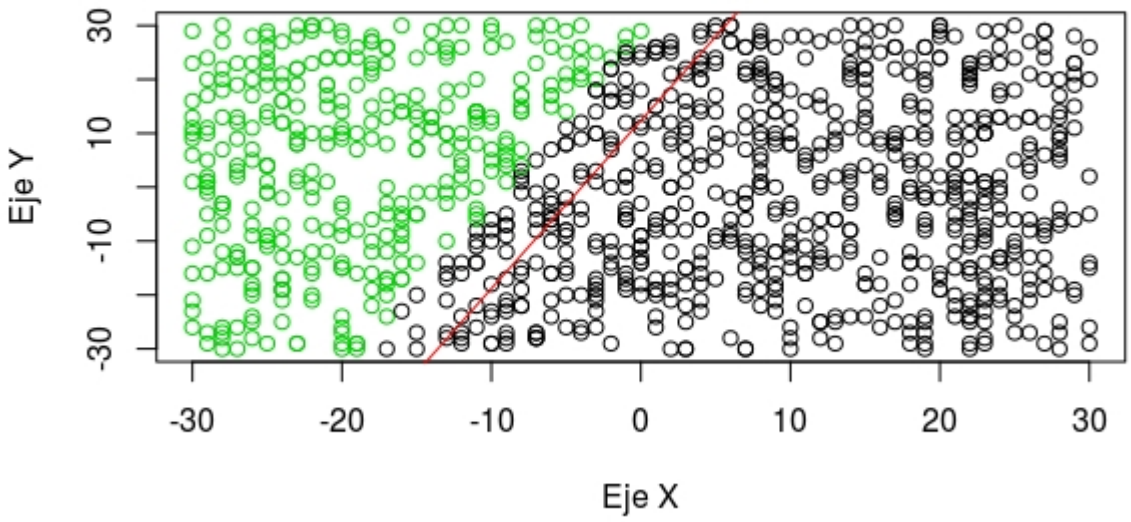

Figura 4.2: Gráficas de regresión logística con $N = 1000$

5. Clasificación de Dígitos. Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 1 y 5. Usar los ficheros de entrenamiento (training) y test que se proporcionan. Extraer las características de intensidad promedio y simetría en la manera que se indicó en el ejercicio 3 del trabajo 1. Plantear un problema de clasificación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g. Usando el modelo de Regresión Lineal para clasificación seguido por PLA-Pocket como mejora. Responder a las siguientes cuestiones.

5.1. Generar gráficos separados (en color) de los datos de entrenamiento y test junto con la función estimada.

La función de regresión lineal es la siguiente:

```
regresion_lineal <- function(datos, signos){

  datos <- datos[,ncol(datos):1]
  datos <- cbind(1,datos)

  # Calculamos la pesudoinversa
  datos.svd <- svd(t(datos)%*%datos)
  U <- datos.svd$u
  V <- datos.svd$v
  D_inversa <- diag(1/datos.svd$d)
  x_pseudo_inver <- (V%*%D_inversa%*%t(U))%*%t(datos)

  # Calculamos los pesos
  w <- x_pseudo_inver%*%digitos

  return(w)
}
```

La función del algoritmo PLA-pocket es:

```
PLA_pocket <- function(datos,label,max_iter,vini){

  pesos_pla <- matrix(vini,nrow=3,ncol=1)
  pesos_pocket <- pesos_pla
```

```
  niter <- 1
  errores <- 0
  errores_ant <- length(datos)+1

  while(niter <= max_iter){
    errores <- 0

    # Ejecutamos PLA para obtener los pesos nuevos
    for(i in 1:nrow(datos)){

      punto <- c(datos[i,],1)
      error <- sum(label[i]-t(pesos_pocket)%*%punto)

      #Actualizamos los pesos
      if(sign(sum(t(pesos_pocket)%*%punto)) != label[i]){
        umbral_ant <- pesos_pocket[length(pesos_pocket)]
        pesos_pla <- pesos_pocket + label[i]*punto
        pesos_pla[length(pesos_pla)] <- umbral_ant + error
        errores <- errores+1
      }
    }

    # Comprobamos si los pesos nuevos son mejores que los
       anteriores
    if(errores < errores_ant){
      pesos_pocket <- pesos_pla
      errores_ant <- errores
    }

    niter <- niter+1
  }

  #Calculamos los coeficientes
  return(pesos_pocket)
}
```

Tras leer los datos de entrenamiento y aplicar los algoritmos de regresión lineal y PLA-pocket el resultado es el siguiente:

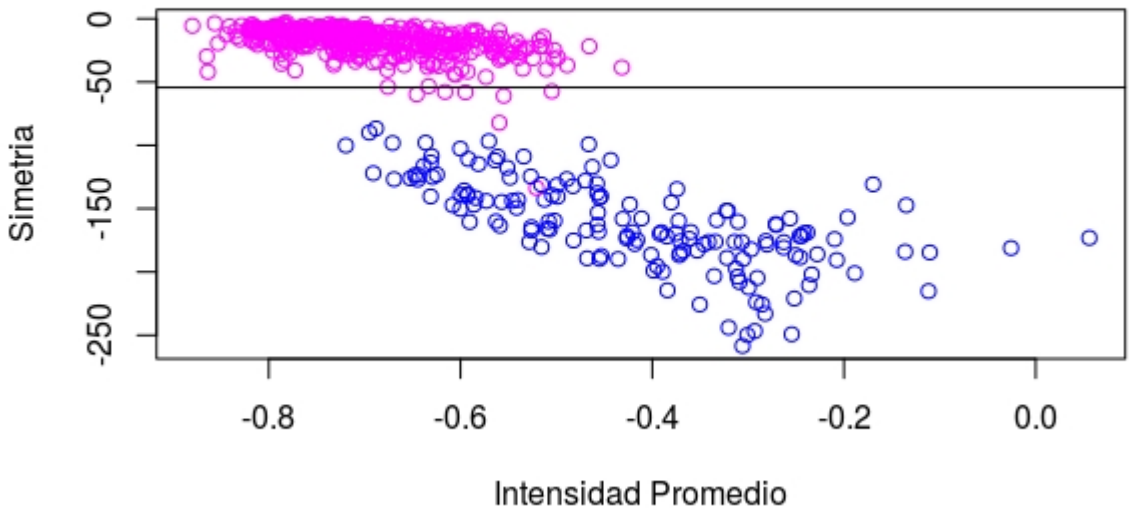

Figura 5.1: Gráfica del conjunto de entrenamiento

Con los datos de test el resultado aplicando la función encontrada con los datos de entrenamiento el resultado es el siguiente:

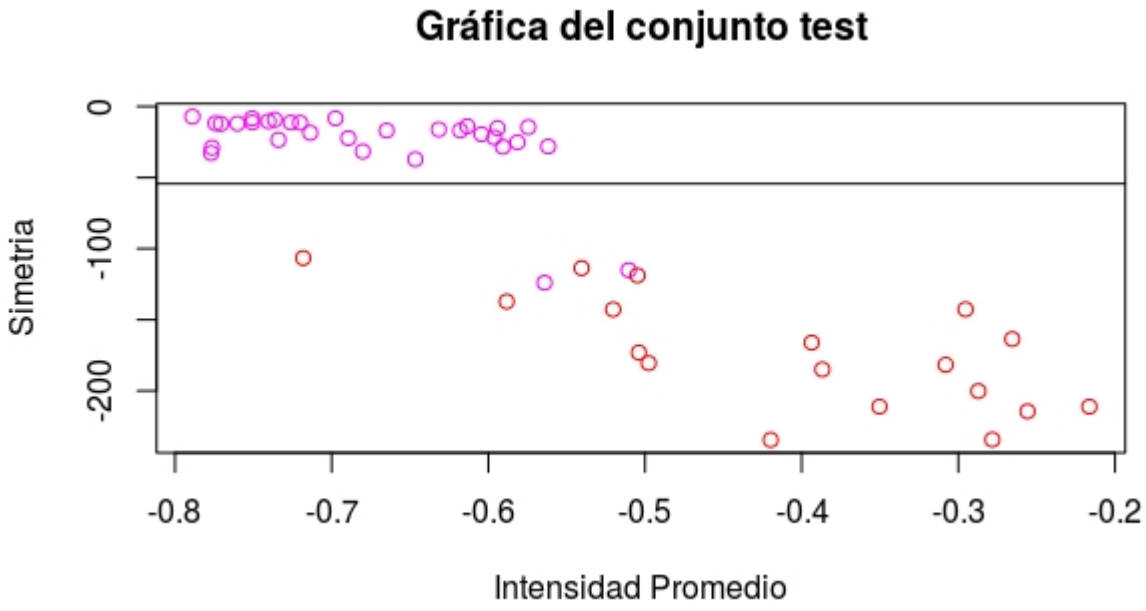

Figura 5.2: Gráfica del conjunto de test

5.2. Calcular E_{in} y E_{test} (error sobre los datos de test).

El error dentro de la muestra es $E_{in} = 1,1686$ y el error sobre los datos de test $E_{test} = 4,081633$.

5.3. Obtener cotas sobre el verdadero valor de E_{out}. Pueden calcularse dos cotas una basada en E_{in} y otra basada en E_{test}. Usar una tolerancia $\delta = 0,05$. ¿Que cota es mejor?

La cota basada en E_{in} es $E_{out} \leq 0,1906$ y con la cota basada en E_{test} $E_{out} \leq 0,2197$. La cota basada en E_{test} es más representativa dado que se ha hecho sobre el error de datos de validación.