# Práctica 2

*Anabel Gómez Ríos*

## 1. Modelos Lineales

### 1. Gradiente Descendente. Implementar el algoritmo de gradiente descendente.

PREGUNTAR SI CONDICIÓN DE PARADA CUANDO SE MUEVA POCO (DIFERENCIA ENTRE VALORES DE LA FUNCIÓN E) Y CUÁL ES ESE POCO

```
# Algoritmo del gradiente descendente. Le pasamos a la función la función de
# error, su gradiente (que serán funciones), el punto en el que se empieza, la
# tasa de aprendizaje, el número máximo de iteraciones a realizar, y el mínimo
# error al que queremos llegar, en orden.
# Devuelve los valores de la función de error por los que pasa junto con la
# iteración.
gradienteDescendente <- function(ferror, gradiente, pini, tasa, maxiter, tope) {
  w <- pini
  i <- 1
  valoresError <- c(i, ferror(pini[1], pini[2]))
  mostrar <- TRUE
  while (i <= maxiter) {
    g <- gradiente(w[1], w[2])
    # Le cambiamos la dirección al gradiente para ir hacia abajo
    v <- -g
    # Nos movemos tanto como indique la tasa
    w <- w + tasa*v
    valoresError <- rbind(valoresError, c(i, ferror(w[1], w[2])))

    if (((abs(ferror(w[1], w[2])) < tope) || (i==maxiter)) && mostrar) {
      cat("He necesitado", i, "iteraciones para llegar al error", ferror(w[1], w[2]),"\n")
      cat("con valores de u y v:", w[1],",", w[2])
      mostrar <- FALSE
    }
    i <- i+1
  }
  return(valoresError)
}
```

**a) Considerar la función no lineal de error $E(u,v) = (ue^v - 2ve^{-u})^2$. Usar gradiente descendente y minimizar esta función de error, comenzando desde el punto $(u,v) = (1,1)$ y usando una tasa de aprendizaje $\eta = 0.1$**

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

Calculamos el gradiente de $E(u,v)$: $\nabla E(u,v) = (\frac{\partial E}{\partial u}, \frac{\partial E}{\partial v}) = (2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}), 2(ue^v - 2e^{-u})(ue^v - 2e^{-u})) = 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}, ue^v - 2e^{-u})$

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)

```
E <- function(u,v) (u*exp(v) - 2*v*exp(-u))^2
gradE <- function(u,v) {(2*(u*exp(v) - 2*v*exp(-u)))*c(exp(v) + 2*v*exp(-u),
                                                       u*exp(v) - 2*exp(-u))}

val <- gradienteDescendente(E, gradE, c(1,1), 0.1, 20, 10^{-14})
```

```
## He necesitado 10 iteraciones para llegar al error 1.208683e-15
## con valores de u y v: 0.04473629 , 0.02395871
```

3) ¿Qué valores de (u,v) obtuvo en el apartado anterior cuando alcanzó el error de $10^{-14}$

Como podemos ver en la salida por pantalla anterior, el valor de u ha sido 0.04473628 y el de v ha sido 0.02395873

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

1) Usar gradiente descendente para minimizar esta función. Usar como valores iniciales $x_0 = 1$, $y_0 = 1$, la tasa de aprendizaje $\eta = 0.01$ 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 $\eta = 0.1$. Comentar las diferencias.

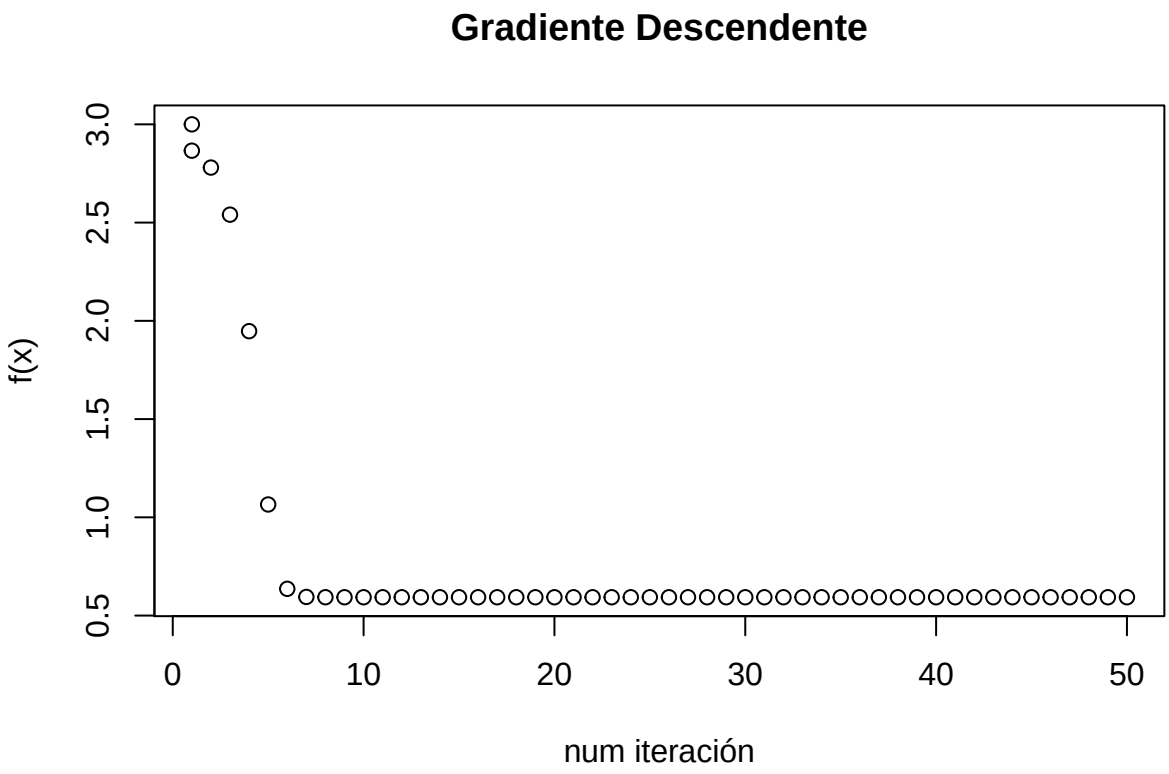
Calculamos primero su gradiente: $\nabla f = (2x + 4\pi * sin(2\pi y) * cos(2\pi x), 4y + 4\pi * sin(2\pi x) * cos(2\pi y))$

```
f <- function(x,y) x^2 + 2*y^2 + 2*sin(2*pi*x)*sin(2*pi*y)
gradF <- function(x,y) c(2*x + 4*pi*sin(2*pi*y)*cos(2*pi*x),
                         4*y + 4*pi*sin(2*pi*x)*cos(2*pi*y))

val <- gradienteDescendente(f, gradF, c(1,1), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 0.5932694
## con valores de u y v: 1.21807 , 0.712812
```

```
plot(val[,1], val[,2], type="p", xlab="num iteración", ylab="f(x)", main="Gradiente Descendente")
```

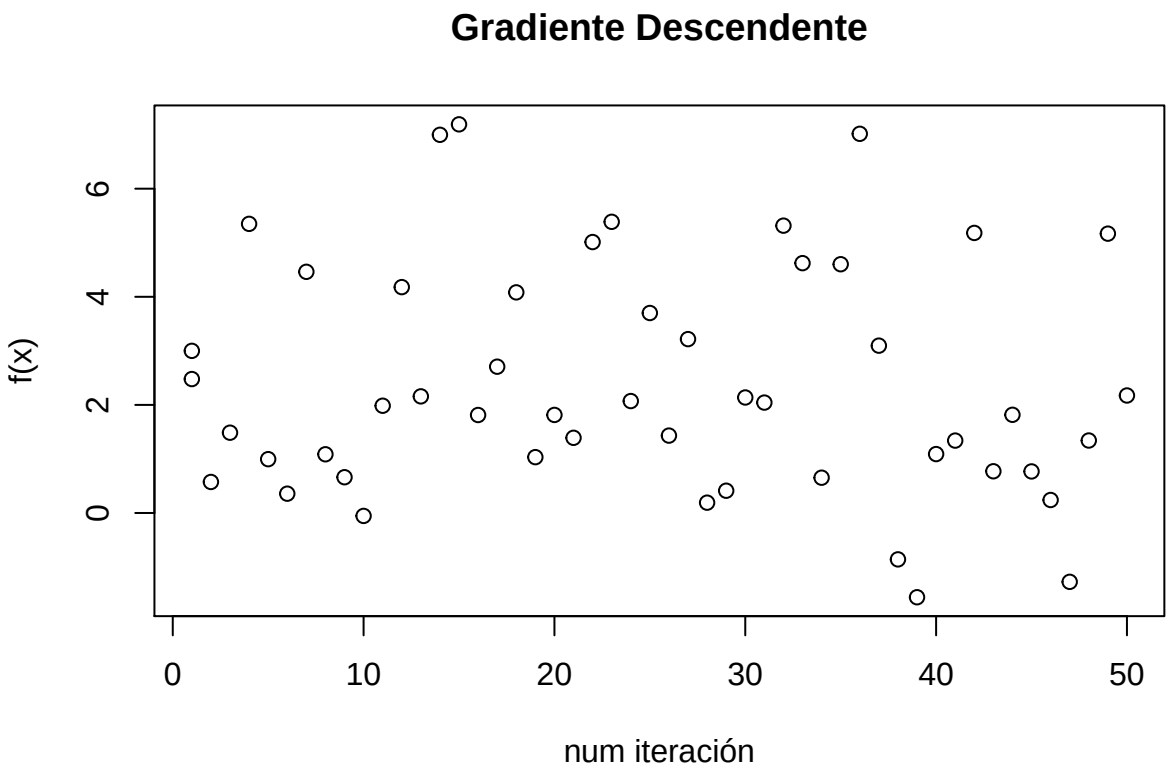



Repetimos con $\eta = 0.1$:

```
val <- gradienteDescendente(f, gradF, c(1,1), 0.1, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 2.173886
## con valores de u y v: 0.6882825 , -0.1732115
```

```
plot(val[,1], val[,2], type="p", xlab="num iteración", ylab="f(x)", main="Gradiente Descendente")
```




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?

PARA QUÉ TASA DE APRENDIZAJE

```
val <- gradienteDescendente(f, gradF, c(0.1,0.1), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error -1.820079
## con valores de u y v: 0.243805 , -0.2379258
```

```
val <- gradienteDescendente(f, gradF, c(1,1), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 0.5932694
## con valores de u y v: 1.21807 , 0.712812
```

```
val <- gradienteDescendente(f, gradF, c(-0.5,-0.5), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error -1.332481
## con valores de u y v: -0.7313775 , -0.2378554
```

```
val <- gradienteDescendente(f, gradF, c(-1,-1), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 0.5932694
## con valores de u y v: -1.21807 , -0.712812
```

**2. Coordenada descendente. En este ejercicio comparamos la eficiencia de la técnica de optimización de "coordenada descendente" usando la misma función del ejercicio 1.1.a. En cada iteración, tenemos dos pasos a lo largo de dos coordenadas. En el paso 1 nos movemos a lo largo de la coordenadas $u$ 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 de $\eta = 0.1$.**

```
# Algoritmo de coordenada descendente. Le pasamos a la función la función de
# error, su gradiente (que serán funciones), el punto en el que se empieza, la
# tasa de aprendizaje, el número máximo de iteraciones a realizar, y el mínimo
# error al que queremos llegar, en orden.
coordenadaDescendente <- function(ferror, gradiente, pini, tasa, maxiter, tope) {
  w <- pini
  i <- 1
  mostrar <- TRUE
  while (i <= maxiter) {
    # Paso 1
    g <- gradiente(w[1], w[2])
    # Le cambiamos la dirección al gradiente para ir hacia abajo
    v <- -g
    w[1] <- w[1] + tasa*v[1]

    # Paso 2
    g <- gradiente(w[1], w[2])
    # Le cambiamos la dirección al gradiente para ir hacia abajo
    v <- -g
    w[2] <- w[2] + tasa*v[2]

    if (((abs(ferror(w[1], w[2])) < tope) || (i==maxiter)) && mostrar) {
      cat("He necesitado", i, "iteraciones para llegar al error", ferror(w[1], w[2]),"\n")
      cat("con valores de u y v:", w[1],",", w[2])
      mostrar <- FALSE
    }
    i <- i+1
  }
}
```

**a) ¿Qué error $E(u, v)$ se obtiene después de 15 iteraciones completas (i.e. 30 pasos)?**

```
val <- coordenadaDescendente(E, gradE, c(1,1), 0.1, 15, 0)
```

```
## He necesitado 15 iteraciones para llegar al error 0.1398138
## con valores de u y v: 6.297076 , -2.852307
```

**b) Establezca una comparación entre esta técnica y la técnica de gradiente descendente.**

POR HACER

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

PREGUNTAR SI ESTO ESTÁ BIEN

```
# Algoritmo del método de Newton. Le pasamos a la función la función de
# error, su gradiente y la matriz hessiana (que serán funciones), el punto
# en el que se empieza, la tasa de aprendizaje, el número máximo de iteraciones a
# realizar, y el mínimo error al que queremos llegar, en orden.
# Devuelve los valores de la función de error por los que pasa junto con la
# iteración.
metodoNewton <- function(ferror, gradiente, hessiana, pini, tasa, maxiter, tope) {
  w <- pini
  i <- 1
  valoresError <- c(i, ferror(pini[1], pini[2]))
  mostrar <- TRUE
  while (i <= maxiter) {
    hg <- solve(hessiana(w[1], w[2]))%*%gradiente(w[1], w[2])
    # Le cambiamos la dirección al gradiente para ir hacia abajo
    v <- -hg
    # Nos movemos tanto como indique la tasa
    w <- w + tasa*v
    valoresError <- rbind(valoresError, c(i, ferror(w[1], w[2])))

    if (((abs(ferror(w[1], w[2])) < tope) || (i==maxiter)) && mostrar) {
      cat("He necesitado", i, "iteraciones para llegar al error", ferror(w[1], w[2]),"\n")
      cat("con valores de u y v:", w[1],",", w[2])
      mostrar <- FALSE
    }
    i <- i+1
  }
  return(valoresError)
}
```

Calculamos la matriz hessiana de la f. Recordemos que las derivadas parciales cruzadas (de existir y ser continuas, como es nuestro caso) son iguales, por el teorema de Schwarz.

```
f <- function(x,y) x^2 + 2*y^2 + 2*sin(2*pi*x)*sin(2*pi*y)
gradF <- function(x,y) c(2*x + 4*pi*sin(2*pi*y)*cos(2*pi*x),
                         4*y + 4*pi*sin(2*pi*x)*cos(2*pi*y))
d12 <- function(x,y) 8*pi^2*cos(2*pi*y)*cos(2*pi*x)
d11 <- function(x,y) 2 - 8*pi^2*sin(2*pi*y)*sin(2*pi*x)
d22 <- function(x,y) 4 - 8*pi^2*sin(2*pi*x)*sin(2*pi*y)
```

```
hess <- function(x,y) rbind(c(d11(x,y), d12(x,y)), c(d12(x,y), d22(x,y)))

val <- metodoNewton(f, gradF, hess, c(1,1), 0.01, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 2.937804
## con valores de u y v: 0.9803919 , 0.9904148
```

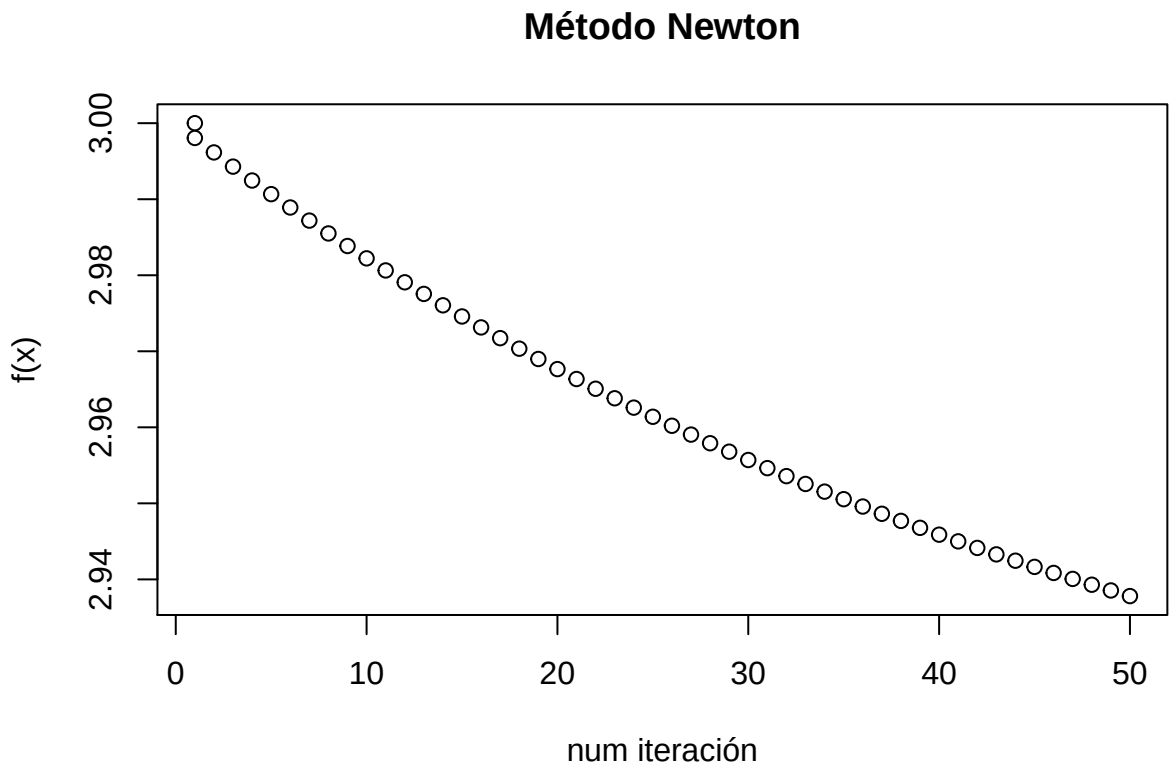
```
val2 <- metodoNewton(f, gradF, hess, c(1,1), 0.1, 50, 0)
```

```
## He necesitado 50 iteraciones para llegar al error 2.900408
## con valores de u y v: 0.9494086 , 0.9747153
```

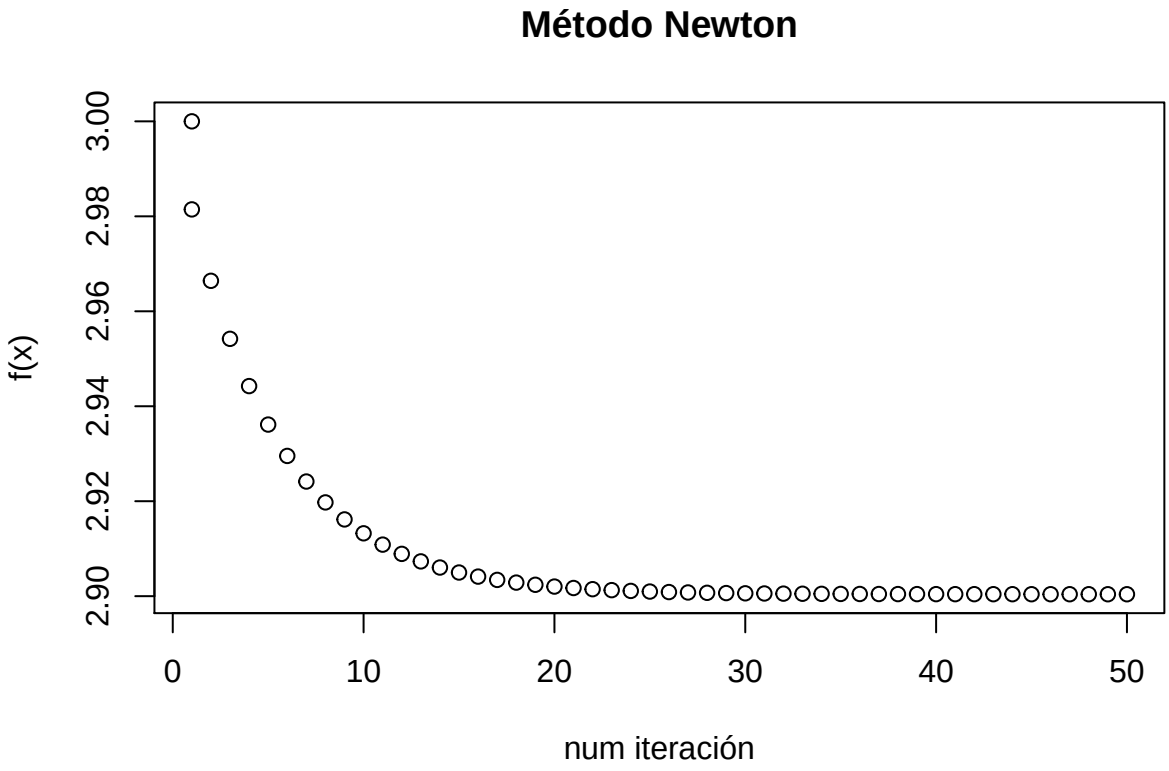
ES ESTO O LO DE DESPUÉS TAMBIÉN?

**a) Generar un gráfico de cómo desciende el valor de la función con las iteraciones.**

```
plot(val[,1], val[,2], type="p", xlab="num iteración", ylab="f(x)", main="Método Newton")
```




```
plot(val2[,1], val2[,2], type="p", xlab="num iteración", ylab="f(x)", main="Método Newton")
```




**b) 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.**

POR HACER

**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 $X = [-1, 1] \times [-1, 1]$ 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.**

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

1. Inicializar el vector de pesos con valores 0.
2. Parar el algoritmo cuando $||w^{(t-1)} - w^{(t)}|| < 0.01$, 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.
3. Aplicar una permutación aleatoria de 1,2,...,$N$ a los datos antes de usarlos en cada época del algoritmo.
4. Usar una tasa de aprendizaje de $\eta = 0.01$.

**b) Usar la muestra de datos etiquetada para encontrar $g$ y estimar $E_{out}$ (el error de entropía cruzada) usando para ello un número suficientemente grande de nuevas muestras.**

QUÉ NARICES ES EL ERROR DE ENTROPÍA CRUZADA

**c) Repetir el experimento 100 veces con diferentes funciones frontera y calcule el promedio.**

1) ¿Cuál es el valor de $E_{out}$ para $N = 100$?
2) ¿Cuántas épocas tada en promedio RL en converger para $N = 100$, usando todas las condiciones anteriormente especificadas?