Trabajo2

Gustavo Rivas Gervilla

MODELOS LINEALES

1. Gradiente descendente Implementar el algoritmo de gradiente descendente.

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 en primer lugar las derivadas parciales de la función respecto a de las variables u y v, que serán las dos componentes del gradiente de la función:

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

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

El gradiente de esta función es el siguiente:

$$\nabla E(u,v) = 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u}, ue^v - 2e^{-u})$$

entonces definimos tanto la función de error como el gradiente de la función para poder usarlo en los métodos que iremos implementando a lo largo de la práctica.

```
E <- function(x) {
  u = x[1]
  v = x[2]
  (u*exp(v)-2*v*exp(-u))**2
}

gradE <- function(x) {
  u = x[1]
  v = x[2]
  2*(u*exp(v)-2*v*exp(-u))*c(exp(v)+2*v*exp(-u), u*exp(v)-2*exp(-u))
}
```

2) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de $E(u,v)$ inferior a 10^{-14}.

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

Nosotros para este algoritmo así como otro que implementaremos más adelante usaremos distintos criterior de parada:

- Que la función alcance o quede por debajo del mínimo establecido.
- Que la distancia entre dos valores consecutivos calculados de f sea menor que una cierta tolerancia.
- Que la distancia entre los vectores de pesos calculados sea menor que dicha tolerancia.

- Que se alcance el máximo de iteracines permitidas.

Entonces aquí mostramos tanto la implementación del algoritmo de gradiente descendente como la de una función para calcular la distancia entre vectores, en lugar de usar la función *dist* que tiene R que calcula distancias entre filas de una matriz.

```
d <- function(x,y) {
  sqrt(sum((x-y)**2))
}

gradDesc <- function(f, gradf, eta, w0, tol, max_iter = 1000000, dibujar = FALSE) {
  w = w0 #inicializamos pesos
  n_iters = 1
  valores_f = NULL

  repeat {
    gt = gradf(w) #calculamos el gradiente
    vt = -gt
    w_ant = w
    f_ant = f(w)
    valores_f = c(valores_f, f_ant)
    w = w + eta*vt #actualizamos el vector de pesos
    n_iters = n_iters+1

    #cond. de parada
    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol){
      valores_f = c(valores_f, f(w))
      break
    }
  }

  cat("Num de iters empleado", n_iters, "\n")
  cat("Valor de f alcanzado", f(w), "\n")
  cat("w obtenido", w, "\n")

  if (dibujar)
    plot(seq(n_iters), valores_f, type = "l", ylab = "Valor de f", xlab = "Iteraciones")
}
```

Y los resultados que obtenemos son:

```
## Num de iters empleado 11
## Valor de f alcanzado 1.208683e-15
## w obtenido 0.04473629 0.02395871
```

Si nos fijamos en la función anterior vamos almacenando en *valores_f* los valores de la función que se van alcanzando con los distintos vectores de pesos, esto lo hacemos para poder dibujar una gráfica en la que se muestre cómo evoluciona el valor de la función a medida que avanzan las iteraciones del algoritmo, algo que se nos pide en el siguiente apartado.

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

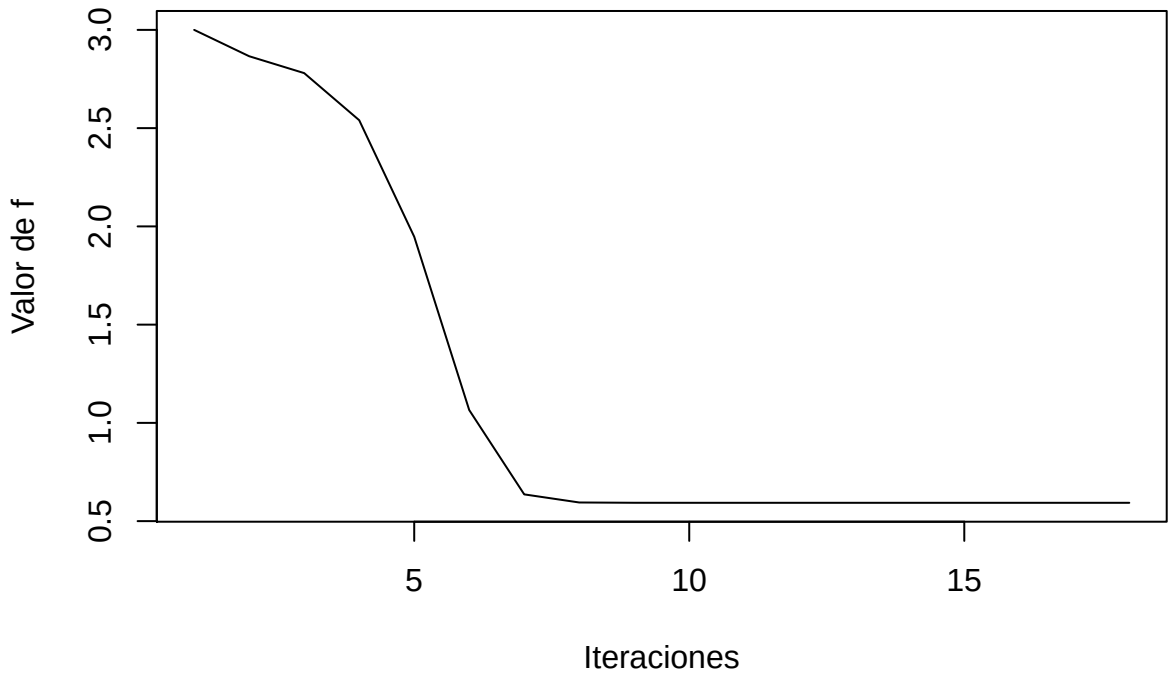
1) Usar gradiente descendente para minimizar esta función. Usar como valores iniciales $x_0 = 1$, $y_0 = 1$, la tasa de aprendizaje $\eta = 0.01$ y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando $\eta = 0.1$, comentar las diferencias.

Definimos la función y su gradiente igual que antes:

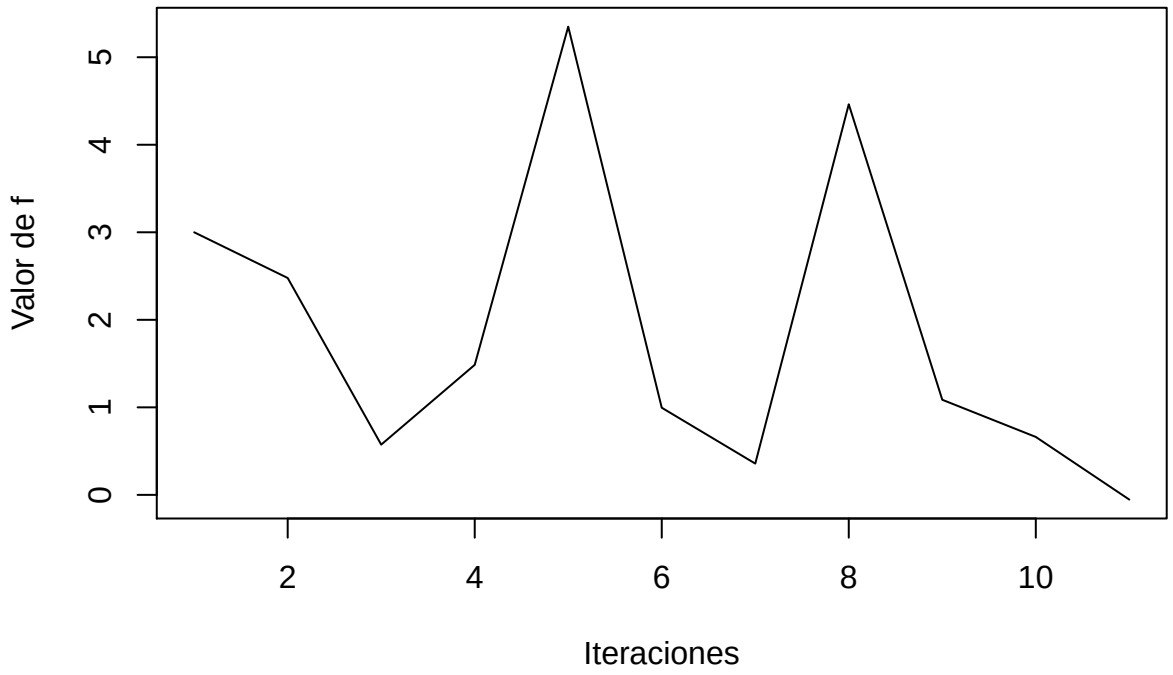
```
f <- function(X){
  x = X[1]
  y = X[2]

  x**2+2*y**2+2*sin(2*pi*x)*sin(2*pi*y)
}

gradf <- function(X) {
  x = X[1]
  y = X[2]

  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))
}
```

y pasamos a los resultados:

```
## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido 1.21807 0.7128119
```




```
## Num de iters empleado 11
## Valor de f alcanzado -0.05388005
## w obtenido 0.3881225 0.6516421
```



Lo que ocurre es que la tasa de aprendizaje nos define la longitud de los "saltos" que damos de un iteración del algoritmo a otra, es decir, el gradiente nos manda la dirección en la que nos movemos pero la tasa de aprendizaje es la que nos dice cuánto nos movemos en dicha dirección. Entonces cuando la tasa de aprendizaje es grande, $\eta = 0.1$, la longitud del "salto" es tan grande que nos pasamos del mínimo, es decir, pasamos a una zona de la función con un valor mayor del que ya estábamos, entonces aunque la convergencia con una tasa de aprendizaje menor es más lenta, es más segura al tener menor probabilidad de tener este efecto.

Lo que si observamos es que, en esta ocasión, llegamos a un valor por debajo del establecido en un menor número de iteraciones puesto que el dar "saltos" de mayor longitud nos ha llevado a una "zona" de la función de valores menores y hemos alcanzado un valor bajo más rápido, pero pensemos por ejemplo en la función x^2 que tiene sólo un mínimo en el cero y que quisiéramos alcanzar dicho mínimo, entonces quizás con una tasa de aprendizaje tan grande no llegaríamos nunca a alcanzarlo.

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?

```
## Num de iters empleado 4
## Valor de f alcanzado -0.0009552139
## w obtenido 0.005266095 -0.002392069

## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido 1.21807 0.7128119

## Num de iters empleado 6
## Valor de f alcanzado -0.01244002
## w obtenido -0.5803234 -0.3839919

## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido -1.21807 -0.7128119
```

Pto. inicial	Mín. alcanzado	Iteraciones Empleadas	Valor de las variables
(0.1,0.1)	-0.0009552139	4	(0.005266095, -0.002392069)
(1,1)	0.5932694	18	(1.21807, 0.7128119)
(-0.5,-0.5)	-0.01244002	6	(-0.5803234, -0.3839919)
(-1,-1)	0.5932694	18	(-1.21807, -0.7128119)

Por como funciona nuestro algoritmo y los criterios de parada que hemos establecido y que hemos mencionado anteriormente tenemos que la principal dificultad es la de partir de un buen punto inicial ya que podemos quedar atrapados en un mínimo local o en una zona de poca variabilidad por eso en los puntos (1,1) y (-1,-1) pese a alcanzar un valor bastante elevado de la función el algoritmo para puesto que la variabilidad entre los distintos valores de la función en iteraciones consecutivas en muy pequeña.

2. Coordenada descendente En este ejercicio compararemos 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 en 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 $\eta = 0.1$.

Veamos la implementación que hemos hecho del algoritmo:

```
coordDesc <- function(f, gradf, eta, w0, tol, max_iter = 1000000) {
  w = w0
  n_iters = 1

  repeat{
    w_ant = w
    f_ant = f(w)
    n_iters = n_iters + 1

    w[1] = w[1] - eta*gradf(w)[1]
    w[2] = w[2] - eta*gradf(w)[2]

    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol) break

  }

  list(w, f(w))
}
```

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

```
## [[1]]
## [1]  6.300845 -2.823852
##
## [[2]]
## [1] 0.1478295
```

El valor que obtenemos es: 0.1478295.

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

Lo que hemos hecho es llamar al gradiente descendente con los mismo parámetros que los empleados para la la llamada a Coordenada Descendente y los resultados obtenido son los siquientes:

```
gradDesc(E, gradE, 0.1, c(1,1), 10^{-14}, 15)
```

```
## Num de iters empleado 11
## Valor de f alcanzado 1.208683e-15
## w obtenido 0.04473629 0.02395871
```

con lo cual como podemos ver la convergencia es mucho más rápida pues emplea menos de 15 iteraciones antes de parar además de alcanzar un valor mucho menor de f. Es lógico que la convergencia sea más rápida ya que estamos cambiando ambas componentes del vector de pesos a la vez.

3. Método de Newton Implementar el algoritmo de Newton y aplicarlo a la función $f(x, y)$ dada en el ejercicio 1b. Desarolle los mismo experimentos usando los mismo 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.

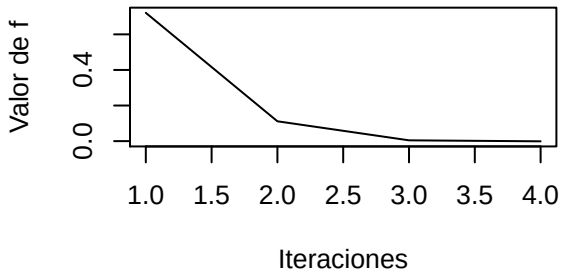
Lo que hacemos antes de definir el método es definir la matriz Hessiana de la función ya que este método hace uso de ella, así tenemos el siguiente código:

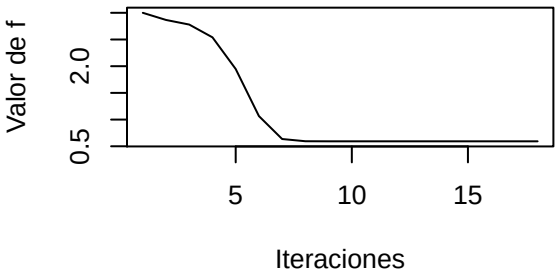
```
Hf <- function(X) {
  x = X[1]
  y = X[2]

  matrix(c(2-8*pi*pi*sin(2*pi*y)*sin(2*pi*x), 8*pi*pi*cos(2*pi*y)*cos(2*pi*x),
           8*pi*pi*cos(2*pi*x)*cos(2*pi*y), 4-8*pi*pi*sin(2*pi*x)*sin(2*pi*y)),
         2,2)
}

Newton <- function(f, gradf, Hf, eta, w0, tol, max_iter = 1000000, dibujar = FALSE){
  w = w0
  n_iters = 1
  valores_f = NULL

  repeat {
    w_ant = w
    f_ant = f(w)

    valores_f = c(valores_f, f_ant)
```

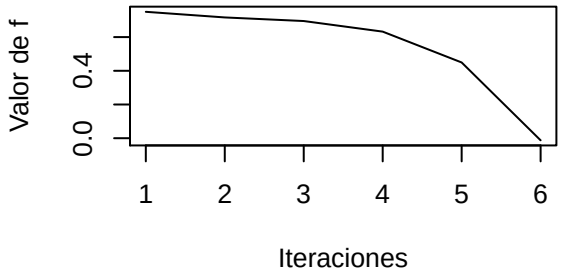
```
    w = w-t(eta*ginv(Hf(w))%*%gradf(w))
    n_iters = n_iters + 1

    if (f(w) < tol || n_iters == max_iter || d(w_ant, w) <= tol || abs(f(w) - f_ant) <= tol){
      valores_f = c(valores_f, f(w))
      break
    }
  }

  cat("Num de iters empleado", n_iters, "\n")
  cat("Valor de f alcanzado", f(w), "\n")
  cat("w obtenido", w, "\n")

  if (dibujar)
    plot(seq(n_iters), valores_f, type = "l", ylab = "Valor de f", xlab = "Iteraciones")

}
```

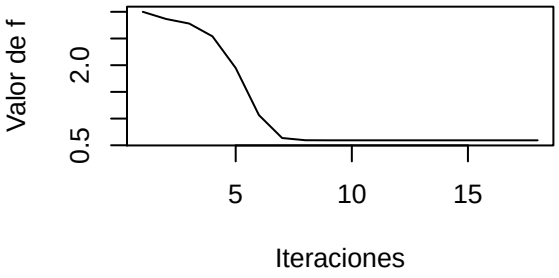
Con gradiente descendente:

```
## Num de iters empleado 4
## Valor de f alcanzado -0.0009552139
## w obtenido 0.005266095 -0.002392069

## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido 1.21807 0.7128119

## Num de iters empleado 6
## Valor de f alcanzado -0.01244002
## w obtenido -0.5803234 -0.3839919

## Num de iters empleado 18
## Valor de f alcanzado 0.5932694
## w obtenido -1.21807 -0.7128119
```



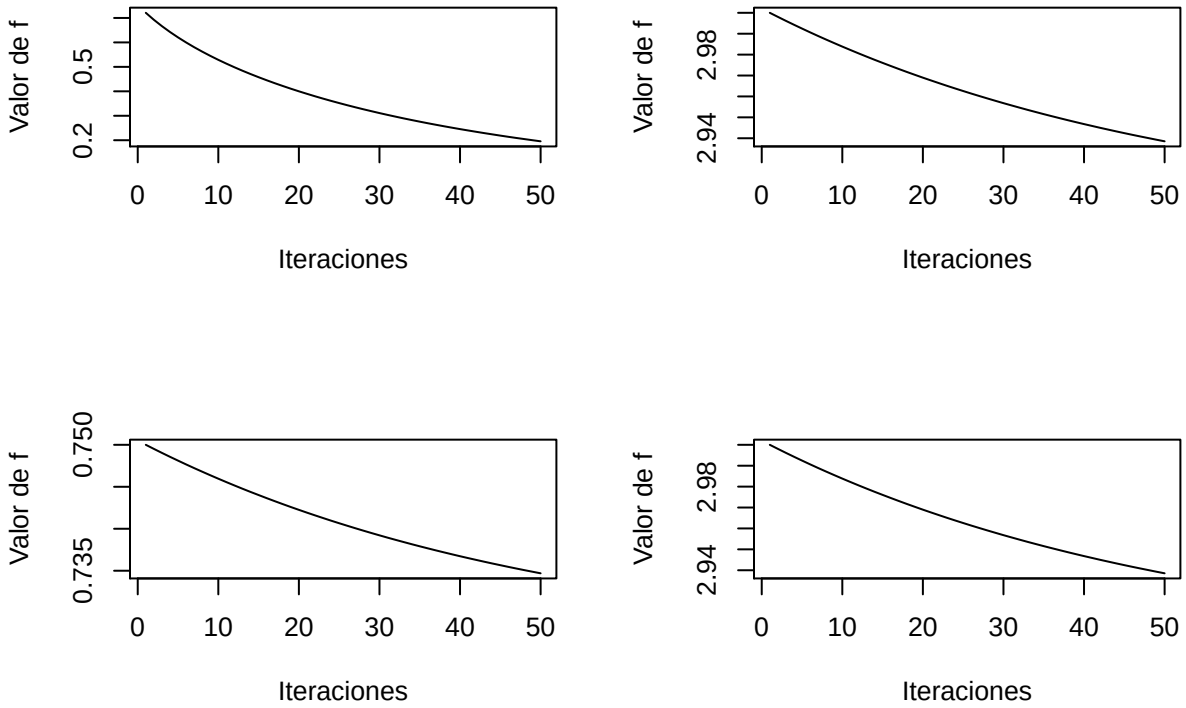






Con el método de Newton:

```
## Num de iters empleado 50
## Valor de f alcanzado 0.1955538
## w obtenido 0.04977112 0.04946853

## Num de iters empleado 50
## Valor de f alcanzado 2.938543
## w obtenido 0.9806977 0.9905654

## Num de iters empleado 50
## Valor de f alcanzado 0.7346939
## w obtenido -0.4903773 -0.4953095

## Num de iters empleado 50
## Valor de f alcanzado 2.938543
## w obtenido -0.9806977 -0.9905654
```



Como podemos ver gradiente descendente nuevamente converge más rápido y obtiene un valor más pequeño de la función, en cambio el método de Newton consume el máximo número de iteraciones permitidas. Lo que sí observamos es como el decrecimiento que experimenta Newton es mucho más suave (también se obtienen más puntos) y siempre desciende. En cambio como vemos el método de gradiente descendente tiene etapas donde se mantiene más o menos contante para luego caer abruptamente y converger.

4. Regresión Logística: En este ejercicio crearemos nuestra propia función objetivo f (probabilística en este caso) y nuestro conjunto de datos $\mathcal{D}$ 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.

Consideramos $d = 2$ para que los datos sean visualizables, y sea $\mathcal{X} = [-1, 1]$x$[-1, 1]$ con probabilidad uniforme de elegir cada $x \in \mathcal{X}$. Elegir una línea en el plano como la frontera entre $f(x) = 1$ (donde y toma valor +1) y $f(x) = 0$ (donde y toma valor -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos.

Seleccionar $N = 100$ punto aleatorios $\{x_n\}$ de $\mathcal{X}$ 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:

- Inicializar el vector de pesos con valores 0.
- 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 una 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 $\eta = 0.01$.

```
RL <- function(datos, et, eta, w0, tol) {
  N = nrow(datos)
  w = w0
  n_etapas = 0

  repeat{
    w_ant = w
    n_etapas = n_etapas + 1

    for (i in sample(N)) {
      gt = (-et[i]*datos[i,])/(1+exp(et[i]*w%*%datos[i,]))
      w = w - eta*gt
    }

    if (d(w, w_ant) < tol)
      break
  }

  list(w, n_etapas)
}
```

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

c) *(Opcional 0.2)* 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 tarda en promedio RL en converger para $N = 100$, usando todas las condiciones anteriormente especificadas?

```
## El número medio de etapas que consume RL antes de converger es:  142.41
## El error medio fuera de la muestra para RL es:  22.33
```

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.

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

b) Obtener E_{in} y E_{test} (error sobre los datos de test).

c) 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 $\delta = 0.05$. ¿Qué cota es mejor?

Tenemos que cuando estamos trabajando con el error dentro de la muestra de entrenamiento la cota que tenemos es la siguiente:

$$E_{out} \leq \sqrt{\frac{8}{N} ln\left(\frac{4(2N^{d_{VC}}+1)}{\delta}\right)}$$

en nuestro caso δ es 0.05 y N es el número de datos de entrenamiento de los que disponemos. Por otro lado como estamos trabajando con hiperplanos de dimensión 2, rectas, tenemos que la d_{VC} es 3.

Por otro lado como podemos ver en el libro Learning from data, si trabajamos con E_{test} entonces podemos emplear sin más la desigualdad de Hoeffding:

$$\mathbb{P}\left(|E_{out} - E_{test}| \geq \varepsilon\right) \leq 2e^{-2\varepsilon^2 N}$$

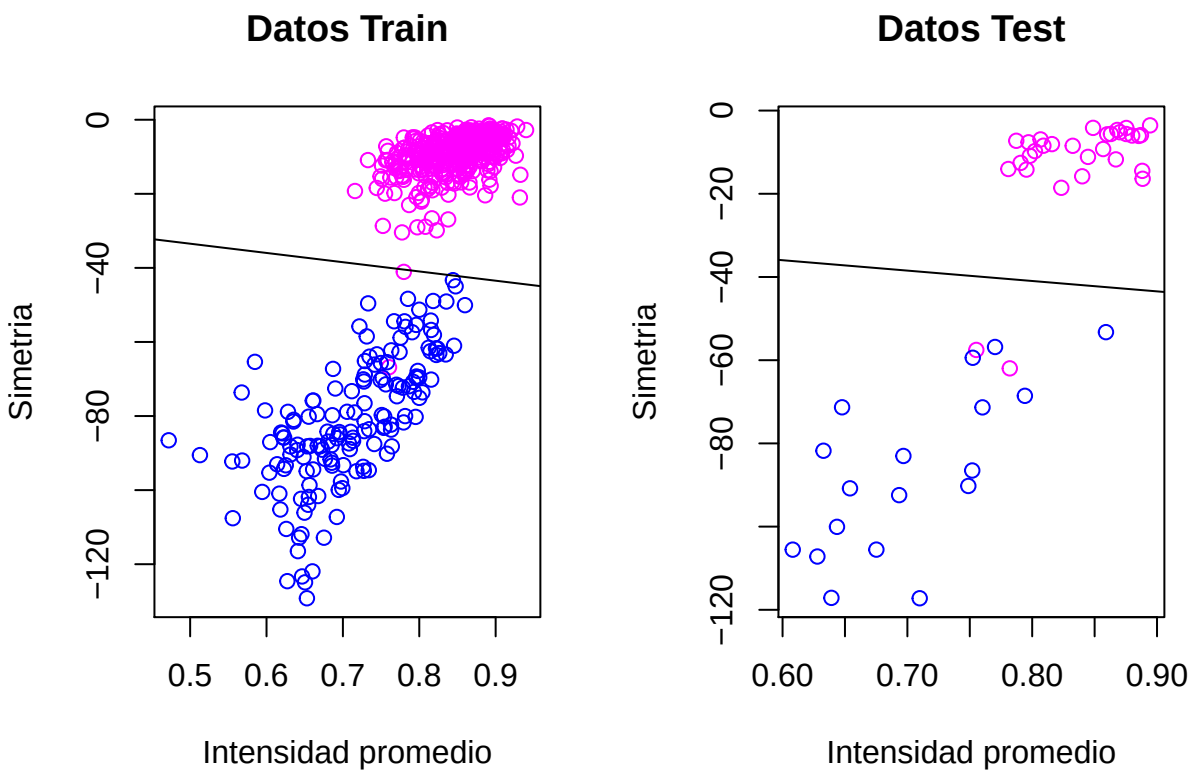
donde ahora N es el número de datos de entrenamiento que tengamos y el ε será aquel que obtengamos al imponer que el término derecho de la desigualdad sea 0.05, la tolerancia que buscamos.

```
## Warning in scan(file, what, nmax, sep, dec, quote, skip, nlines,
## na.strings, : número de items leídos no es múltiplo del número de columnas

## Warning in scan(file, what, nmax, sep, dec, quote, skip, nlines,
## na.strings, : número de items leídos no es múltiplo del número de columnas
```

```
ejercicio1.5 <- function(){

  instancias_train <- datos_train[(datos_train[,1] == 1 | datos_train[,1] == 5),]
  instancias_test <- datos_test[(datos_test[,1] == 1 | datos_test[,1] == 5),]

  m_datos_train <- data.matrix(instancias_train)
  m_datos_test <- data.matrix(instancias_test)

  m_datos_train[,2:257] <- 0.5*(1-m_datos_train[,2:257])
  m_datos_test[,2:257] <- 0.5*(1-m_datos_test[,2:257])
```

```
  et_train <- m_datos_train[,1]
  et_train[et_train == 5] = -1
  et_test <- m_datos_test[,1]
  et_test[et_test == 5] = -1

  imagenes_train <- list()
  imagenes_test <- list()

  for (i in seq(1, nrow(m_datos_train)))
    imagenes_train <- c(imagenes_train, list(getImagen(m_datos_train[i,2:257])))

  for (i in seq(1, nrow(m_datos_test)))
    imagenes_test <- c(imagenes_test, list(getImagen(m_datos_test[i,2:257])))

  medias_train <- unlist(lapply(imagenes_train, mean))
  simetrias_train <- unlist(lapply(imagenes_train, getSimetria))

  medias_test <- unlist(lapply(imagenes_test, mean))
  simetrias_test <- unlist(lapply(imagenes_test, getSimetria))

  datos <- cbind(medias_train, simetrias_train, 1)
  datos_test <- cbind(medias_test, simetrias_test, 1)

  w_ini <- t(regressLin(datos, et_train)) #calculamos el vector de pesos dado por la regresión
  w <- ajusta_PLA_MOD(datos, et_train, 100, w_ini)[[1]] #medimos el numero medio de iteraciones

  Y_wlin_train = apply(datos, 1, h, w = w) #etiquetas dadas por el vector de pesos
  Y_wlin_test = apply(datos_test, 1, h, w = w) #etiquetas dadas por el vector de pesos

  E_in = length(which(et_train != Y_wlin_train))
  E_test = length(which(et_test != Y_wlin_test))
  cat("El error dentro de la muestra es, E_in: ", E_in, "\n")
  cat("El error fuera  de la muestra es, E_test: ", E_test, "\n")

  par(mfrow = c(1,2))
  plot(medias_train, simetrias_train, col = et_train+5, xlab = 'Intensidad promedio', ylab = 'Simetria'
  plotw(w)

  plot(medias_test, simetrias_test, col = et_test+5, xlab = 'Intensidad promedio', ylab = 'Simetria', ma
  plotw(w)


  N_train = length(medias_train) #numero de datos de entrenamiento que tenemos
  N_test = length(medias_test) #numero de dato de test que tenemos

  cat("Usando E_in tenemos que Eout <= ", E_in + sqrt((8/N_train)*log((4*(2*N_train^3+1))/0.05)), "\n" )

  eps = sqrt(log(0.05/2)/(-2*N_test))
  cat("Usando E_test tenemos que |E_test - E_out| <=", eps, "\n")
}
```

```
## El error dentro de la muestra es, E_in:  1
## El error fuera  de la muestra es, E_test:  2
```




```
## Usando E_in tenemos que Eout <=  1.569227
## Usando E_test tenemos que |E_test - E_out| <= 0.1940145
```

Entonces tenemos que nosotros obtenemos mejor cota empleando E_{in}.

SOBREAJUSTE

1. Sobreajuste. Vamos a construir un entorno que nos permita experimentar con los problemas de sobreajuste. Consideremos el espacio de entrada $\mathcal{X} = [-1, 1]$ con una densidad de probabilidad uniforme, $\mathbb{P} = \frac{1}{2}$. Consideramos dos modelos $\mathcal{H}_2$ y $\mathcal{H}_{10}$ representando el conjunto de todos los polinomios de grado 2 y grado 10 respectivamente. La función objetivo es un polinomio de grado Q_f que escribimos como $f(x) = \sum_{q=0}^{Q_f} a_q L_q(x)$, donde $L_q(x) son los polinomio sde Legendre. El conjunto de datos es $\mathcal{D} = \{(x_1, y_1), ..., (x_N, y_N)\}$ donde $y_n = f(x_n) + \sigma\varepsilon_n$ y las $\{\varepsilon_n\}$ son variables aleatorias i.i.d. $\mathcal{N}(0, 1)$ y σ^2 la varianza del ruido.

Comenzamos realizando un experimento donde suponemos que los valores $Q_f, N, \sigma,$ están especificados, para ello:

- Generamos los coeficientes a_q a partir de muestras de una distribución $\mathcal{N}(0, 1)$ y escalamos dichos coeficientes de manera que $\mathbb{E}_{a,x}[f^2] = 1$. (Ayuda: Dividir los coeficientes por $\sqrt{\sum_{q=0}^{Q_f} \frac{1}{2q+1}}$).
- Generamos un conjunto de datos $x_1, ..., x_N$ muestreando de forma independiente (x) y los valores $y_n = f(x_n) + \sigma\varepsilon_n$.

Sean g_2 y g_{10} los mejores ajustes a los datos usando $\mathcal{H}_2$ y $\mathcal{H}_{10}$ respectivamente, y sean $E_{out}(g_2)$ y $E_{out}(g_{10})$ sus respectivos errores fuera de la muestra.

a) Calcular g_2 y g_{10}

En primer lugar veamos las funciones que definimos para calcular de una forma recursiva y eficiente el valor de un determinado polinomio de Legendre así como un valor de la función f que es nuestra función objetivo:

```
#función recursiva para calcular el valor de un Polinomio de Legendre
LegendreRecursivo <- function(x, grado) {
  if (grado == 0)
    return(c(1,1))
  else if (grado == 1)
    return(c(x,1))
  else
    n = grado-1
    componentes = LegendreRecursivo(x, n)
    mi_valor = (2*n+1)/(n+1)*x*componentes[1]-n/(n+1)*componentes[2]
    return(c(mi_valor, componentes[1]))
}

#función para obtener el valor de un polinomio de Legendre sin tener que usar el vector devuelto por la
Legendre <- function(x, grado) {
  LegendreRecursivo(x, grado)[1]
}

fL <- function(x, coef) {
  Qf = length(coef)
  sum(coef*unlist(lapply(seq(0,Qf-1), Legendre, x = x)))
}
```

```
ejercicio2.1 <- function() {
  #damos valores a los parámetros del experimento
  Qf = 20
  N = 50
  sigma = 1

  #generamos los coeficientes de la función f
  term_normalizacion = sqrt(sum(1/(2*seq(0,Qf)+1)))
  coef = runif(Qf)/term_normalizacion

  #generamos los datos
  X = runif(N, -1, 1)
  ruido = rnorm(N)
  Y = unlist(lapply(X, fL, coef = coef)) + sigma*ruido

  Mgrado2 = matrix(X**rep(seq(0,2), each = N), nrow = N)
  Mgrado10 = matrix(X**rep(seq(0,10), each = N), nrow = N)

  w2 = regressLin(Mgrado2, Y)
  w10 = regressLin(Mgrado10, Y)

  cat("g2 ", t(w2), "\n")
```

```
  cat("g10", t(w10), "\n")
}
```

```
## g2  -0.1827841 -0.1657208 1.106736
## g10 -0.401873 -1.806999 -8.480163 -16.9614 119.3675 128.2108 -383.4549 -235.7379 464.6776 129.224 -19
```

b) ¿Por qué normalizamos f? (Ayuda: interpretar el significado de σ).

Dado que estamos muestreando el ruido en el intervalo [-1,1] el sigma nos dice la verdadera distorsión o el "coeficiente de distorsión" real que tenemos en los datos, ya que ahora el ruido estará en el intervalo $[-\sigma, \sigma]$ con lo que a más sigma más distorsión tendremos en los datos.

En el libro Learning from data podemos ver que esta normalización lo que hace es que el nivel de ruido se equilibre al ruido de la señal f, en mi opinión lo que estamos haciendo al normalizar la f es que este ruido no sea tan grande, o al menos que la distorsión sea menor al estar "restringiendo" los valores que puede tomar la f, de este modo paliamos un poco el efecto del ruido.

2. Siguiendo con el punto anterior, usando la combinación de parámetros $Q_f = 20, N = 50, \sigma = 1$ ejecutar un número de experimentos (> 100) calculando en cada caso $E_{out}(g_2)$ y $E_{out}(g_{10})$. Promediar todos los valores de error obtenidos para cada conjunto de hipótesis, es decir,

$E_{out}(\mathcal{H}_2)$ = promedio sobre experimentos ($E_{out}(g_2)$).

$E_out(\mathcal{H}_{10})$ = promedio sobre experimentos ($E_{out}(g_{10})$).

Definimos una medida de sobreajuste como $E_out(\mathcal{H}_{10}) - E_out(\mathcal{H}_2)$.

a) Argumentar por qué la medida dada puede medir el sobreajuste.

```
ejercicio2.2 <- function() {
  Qf = 20
  N = 50
  sigma = 1

  E_out_total_2 = 0
  E_out_total_10 = 0


  for (i in seq(1,200)) {
    #generamos los coeficientes de la función f
    term_normalizacion = sqrt(sum(1/(2*seq(0,Qf)+1)))
    coef = runif(Qf)/term_normalizacion

    #generamos los datos
    X = runif(N, -1, 1)
    ruido = rnorm(N)
    Y_ruido = unlist(lapply(X, fL, coef = coef)) + sigma*ruido
```

```
    Mgrado2 = matrix(X**rep(seq(0,2), each = N), nrow = N)
    Mgrado10 = matrix(X**rep(seq(0,10), each = N), nrow = N)

    w2 = regressLin(Mgrado2, Y_ruido)
    w10 = regressLin(Mgrado10, Y_ruido)

    #calculamos Eout
    X_out = runif(100, -1,1)
    Y_out = unlist(lapply(X_out, fL, coef = coef))

    Mgrado2_out = matrix(X_out**rep(seq(0,2), each = 100), nrow = 100)
    Mgrado10_out = matrix(X_out**rep(seq(0,10), each = 100), nrow = 100)

    Y_out_w2 = apply(Mgrado2_out, 1, function(x) w2%*%x)
    Y_out_w10 = apply(Mgrado10_out, 1, function(x) w10%*%x)

    E_out_w2 = sum((Y_out - Y_out_w2)^2)/100
    E_out_w10 = sum((Y_out - Y_out_w10)^2)/100

    E_out_total_2 = E_out_total_2 + E_out_w2
    E_out_total_10 = E_out_total_10 + E_out_w10
  }

  cat("El error medio fuera de la muestra para H2 es: ", E_out_total_2/200, "\n")
  cat("El error medio fuera de la muestra para H10 es: ", E_out_total_10/200, "\n")
}
```

```
## El error medio fuera de la muestra para H2 es:  3.339305
## El error medio fuera de la muestra para H10 es:  978783.1
```

Podemos medir con esto el sobreajuste ya que podemos comprobar si aplicando un modelo mucho más complejo obtenemos un error fuera de la muestra mayor que en esencia es lo nos indica que tenemos sobreajuste, en un caso normal tendríamos que con un modelo más grande tendríamos un mejor error fuera de la muestra, pero cuando se da sobreajuste entonces sucede lo contrario, los modelos simples son los que tienen mejor capacidad de generalización.

REGULARIZACIÓN Y SELECCIÓN DE MODELOS

1. Para $d = 3$ (dimensión) generar un conjunto de N datos aleatorios $\{x_n, y_n\}$ de la siguiente forma. Para cada punto x_n generamos coordenadas muestreando de forma independiente una $(N)(0, 1)$. De forma similar generamos un vector de pesos de $(d + 1)$ dimensiones w_f, y el conjunto de valores $y_n = w_f^T x_n + \sigma\epsilon_n$, donde ϵ_n es un ruido que sigue también una $(N)(0, 1)$ y σ^2 es la varianza del ruido; fijar $\sigma = 0.5$. Usar regresión lineal con regularización "weight decay" para estimar w_f con w_{reg}. Fijar el parámetro de regularización a $0.05/N$.

a) Para $N \in \{d + 15, d + 25, ..., d + 115\}$ calcular los errores $e_1, ..., e_N$ de validación cruzada y E_{cv}.

b) Repetir el experimento 10^3 veces, anotando el promedi y la varianza de e_1, e_2 y E_{cv} en todos los experimentos.

c) ¿Cuál debería ser la relación entre el promedio de los valores de e_1 y el de los valores de E_{cv}? ¿y el de los valores de e_2? Argumentar la respuesta en base a los resultados de los experimentos.

Lo que ocurre es que para cada e_i (que al fin y al cabo sólo se diferencian en qué elemento quitamos del conjunto de datos, elementos generados aleatoriamente) se da que $\mathbb{E}_{\mathcal{D}}[e_i] = \bar{E}_{out}(N - 1)$, es decir, el promedio de e_i nos da la media del error fuera de la muestra de la función estimada cuando estamos aprendiendo con conjuntos de datos de tamaño $N - 1$, entonces como esto se da para todos los e_i también se dará para su media, es decir, el E_{cv}. Con lo cual lo que ocurre es que estos promedios serán muy similares entre sí como podemos ver en los resultados de los experimentos.

d) ¿Qué es lo que contribuye a la varianza de los valores de e_1?

e_1 es una medida del error de validación para el "primer conjunto de validación cruzada" creado sobre los datos, entonces la varianza estará influida tanto por las diferentes muestras que se generan como de las etiquetas con ruido (aleatorio) que les asignamos, puesto que e_1 viene determinado por el ajuste que hace la regresión líneal sobre los datos que le damos.

e) Si los errores de validación-cruzada fueran verdaderamente independientes, ¿cuál sería la relación entre la varianza de los valores de e_1 y la varianza de los de E_{cv}?

Para este apartado vamos a usar la siguiente fórmula $Var\left(\sum_{i=1}^{N} a_i X_i\right) = \sum_{i=1}^{N} a_i^2 Var(X_i) + 2\sum_{1 \leq i < j \leq N} a_i a_j Cov(X_i, X_j)$, en nuestro caso como asumimos que son independientes entonces los términos de las covarianzas desaparecen y si tenemos en cuenta que las varianzas de los e_i serán todas iguales entonces lo que tenemos es que:

$$Var(E_{cv}) = \frac{1}{N} Var(e_1)$$

f) Una medida del número efectivo de muestras nuevas usadas en el cálculo de E_{cv} es el cociente entre la varianza de e_1 y la varianza de E_{cv}. Explicar por qué, y dibujar, respecto de N, el número efectivo de nuevos ejemplos (N_{eff}) como un porcentaje de N. NOTA: Debería encontrarse que N_{eff} está cercano a N.

Lo primero que tenemos que hacer es definir la función que realiza la regresión lineal con la condición de regularización weight decay, simplemente siguiendo la fórmula vista en teoría:

```
regressLinWeightDecay <- function(Z, label, lambda) {
  ncol = ncol(Z)
  ginv(t(Z)%*%Z + lambda*diag(ncol))%*%t(Z)%*%label
}
```

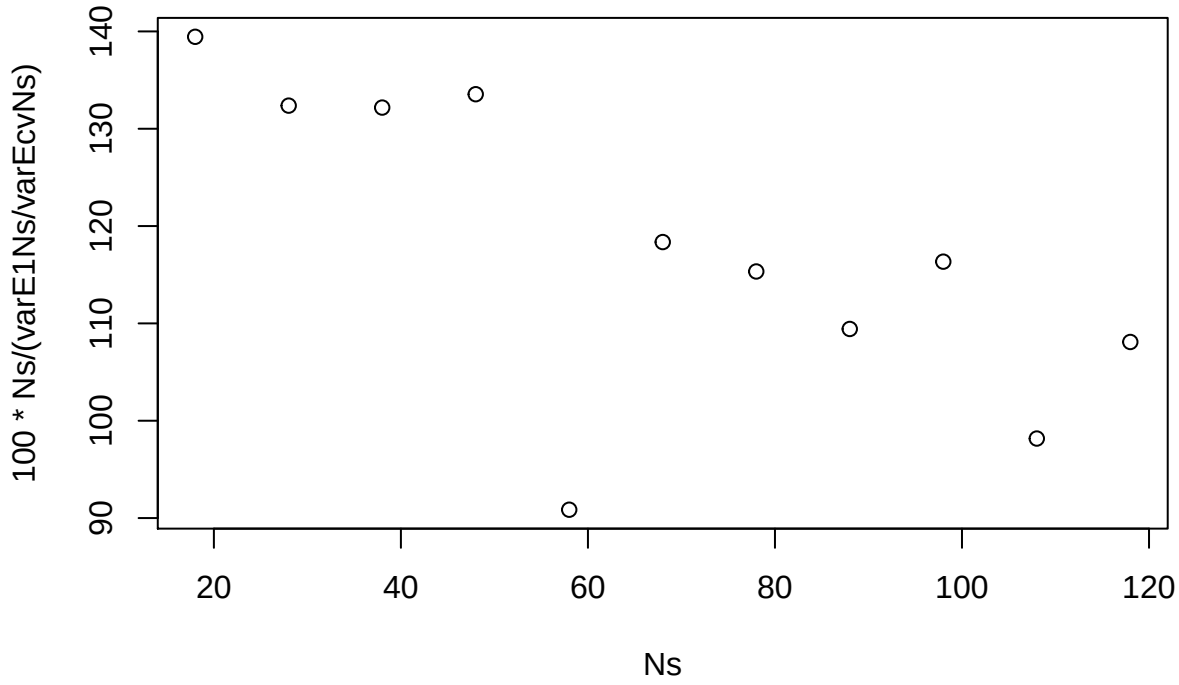
Entonces todos los apartados del ejercicio se tienen en la siguiente función:

```
ejercicio3.1 <- function(term_norm = 0.05) {
  Nexp = 10^3
  d = 3

  Ns = seq(15,115,10)+d
  results = lapply(Ns+1, function(i) rep(0,i))
  wf = rnorm(d+1)

  varE1Ns = rep(0, length(Ns))
  varEcvNs = rep(0,length(Ns))

  for (N in Ns) {
    e1s = rep(0, Nexp)
    e2s = rep(0, Nexp)
    Ecvs = rep(0, Nexp)

    for (i in seq(1,Nexp)) {
      X = simula_gaus(N, d, 1)
      X = cbind(X, 1)
      Y = apply(X, 1, function(x) wf%*%x)
      Y_ruido = apply(X, 1, function(x) wf%*%x + 0.5*rnorm(1))

      idx = seq(1,N)

      Eis <- sapply(idx, function(i) {
        wlin = regressLinWeightDecay(X[-i,], Y_ruido[-i], term_norm/N)
        ei = (t(wlin)%*%X[i,] - Y_ruido[i])^2
      })

      e1s[i] = Eis[1]
      e2s[i] = Eis[2]
      Ecvs[i] = mean(Eis)

      result = c(Eis, mean(Eis))
      results[[N%/%10]] <- results[[N%/%10]] + result/Nexp
    }

    varE1Ns[N%/%10] = var(e1s)
```

```
    varEcvNs[N%/%10] = var(Ecvs)

    cat("Para N: ", N, "e1 medio es ", mean(e1s), " y su varianza ", varE1Ns[N%/%10], " \n")
    cat("Para N: ", N, "e2 medio es ", mean(e2s), " y su varianza ", var(e2s), " \n")
    cat("Para N: ", N, "e1 medio es ", mean(Ecvs), " y su varianza ", varEcvNs[N%/%10], " \n\n\n")
  }

  cat("Los Neff para los distintos N con los que trabajamos:\n ", varE1Ns/varEcvNs, "\n")
  cat("La media de los anteriores valores es: ", mean(varE1Ns/varEcvNs))

  #dibujar número efectivo
  par(mfrow = c(1,1))
  plot(Ns, 100*Ns/(varE1Ns/varEcvNs))
}
```

```
## Para N:  18 e1 medio es  0.3276382  y su varianza  0.2279842
## Para N:  18 e2 medio es  0.3301286  y su varianza  0.2450494
## Para N:  18 e1 medio es  0.3246246  y su varianza  0.01766189
##
##
## Para N:  28 e1 medio es  0.2678827  y su varianza  0.1543628
## Para N:  28 e2 medio es  0.292582  y su varianza  0.176669
## Para N:  28 e1 medio es  0.2934271  y su varianza  0.007297737
##
##
## Para N:  38 e1 medio es  0.270718  y su varianza  0.138272
## Para N:  38 e2 medio es  0.285991  y su varianza  0.1607256
## Para N:  38 e1 medio es  0.2814116  y su varianza  0.004809447
##
##
## Para N:  48 e1 medio es  0.2516532  y su varianza  0.1276315
## Para N:  48 e2 medio es  0.2785182  y su varianza  0.153907
## Para N:  48 e1 medio es  0.2741611  y su varianza  0.003551031
##
##
## Para N:  58 e1 medio es  0.2976852  y su varianza  0.1728054
## Para N:  58 e2 medio es  0.2643783  y su varianza  0.1680089
## Para N:  58 e1 medio es  0.2695083  y su varianza  0.002707017
##
##
## Para N:  68 e1 medio es  0.2575314  y su varianza  0.1334363
## Para N:  68 e2 medio es  0.2690176  y su varianza  0.1659826
## Para N:  68 e1 medio es  0.2680451  y su varianza  0.002322517
##
##
## Para N:  78 e1 medio es  0.2427475  y su varianza  0.1155136
## Para N:  78 e2 medio es  0.2590315  y su varianza  0.1210341
## Para N:  78 e1 medio es  0.2630521  y su varianza  0.001708042
##
##
## Para N:  88 e1 medio es  0.2471435  y su varianza  0.1236896
## Para N:  88 e2 medio es  0.2660944  y su varianza  0.1374411
## Para N:  88 e1 medio es  0.2642761  y su varianza  0.00153806
```

```
##
##
## Para N:  98 e1 medio es  0.2449571  y su varianza  0.1162439
## Para N:  98 e2 medio es  0.2721763  y su varianza  0.1696334
## Para N:  98 e1 medio es  0.2615744  y su varianza  0.001380013
##
##
## Para N:  108 e1 medio es  0.2628195  y su varianza  0.1299504
## Para N:  108 e2 medio es  0.2524828  y su varianza  0.1475563
## Para N:  108 e1 medio es  0.2607238  y su varianza  0.001181229
##
##
## Para N:  118 e1 medio es  0.2547049  y su varianza  0.1298677
## Para N:  118 e2 medio es  0.2629656  y su varianza  0.1475705
## Para N:  118 e1 medio es  0.2583371  y su varianza  0.001189629
##
##
## Los Neff para los distintos N con los que trabajamos:
##   12.90826 21.15215 28.75008 35.94209 63.8361 57.45332 67.62923 80.4192 84.23391 110.0129 109.1666
## La media de los anteriores valores es:  61.0458
```



Es claro por lo que hemos dicho antes que en caso de que los e_i fuesen independientes entonces se tendría que este estimador da N, que como podemos ver en el libro Learning from data es el extremo superior para este estimador (siendo el menor 1), con lo cual parece indicar que estamos en el camino correcto.

En mi opinión esto establece un buen estimador ya que en caso de que hubiese relación entre los distintos errores entonces se tendría que en la fórmula que hemos dado antes para la varianza de E_{cv} no desaparecerían todos los términos de varianza, haciendo este valor más pequeño (suponiendo que las varianzas sean positivas). De algún modo el estimador mide cómo participa la varianza de e_1 en la de E_{cv} en caso de no ser independientes entonces no participa tanto puesto que "unas varianzas se podrían solapar con otras", en cambio como hemos dicho en caso de ser independientes sí que se tiene el máximo valor posible.

g) Si se incrementa la cantidad de regularización, ¿debería N_{eff} subir o bajar? Argumentar la respuesta. Ejecutar el mismo experimento con $\lambda = 2.5/N$ y comparar los resultados del punto anterior para verificar la conjetura.

Dado que el hecho de hacer una regresión con una condición extra como es la de weight decay nos hace mantener un compromiso entre disminuir el error dentro de la muestra y el peso de $\lambda w^T w$ entonces parece lógico pensar que en esta ocasión los e_i serán mayores, con lo que sus varianzas también lo serán haciendo que el $Neff$ sea mayor.

```
## Para N:  18 e1 medio es  0.335936  y su varianza  0.2533542
## Para N:  18 e2 medio es  0.2988058  y su varianza  0.1793787
## Para N:  18 e1 medio es  0.3269673  y su varianza  0.01656963
##
##
## Para N:  28 e1 medio es  0.2845052  y su varianza  0.1642458
## Para N:  28 e2 medio es  0.2952834  y su varianza  0.1832829
## Para N:  28 e1 medio es  0.2933652  y su varianza  0.006814107
##
##
## Para N:  38 e1 medio es  0.2823564  y su varianza  0.1641934
## Para N:  38 e2 medio es  0.2931173  y su varianza  0.1831123
## Para N:  38 e1 medio es  0.2758782  y su varianza  0.00447904
##
##
## Para N:  48 e1 medio es  0.2660964  y su varianza  0.1353655
## Para N:  48 e2 medio es  0.2720847  y su varianza  0.1433795
## Para N:  48 e1 medio es  0.2714197  y su varianza  0.0033425
##
##
## Para N:  58 e1 medio es  0.259894  y su varianza  0.1403824
## Para N:  58 e2 medio es  0.2708537  y su varianza  0.1449147
## Para N:  58 e1 medio es  0.2697423  y su varianza  0.002598311
##
##
## Para N:  68 e1 medio es  0.2370302  y su varianza  0.1181602
## Para N:  68 e2 medio es  0.2701761  y su varianza  0.1330153
## Para N:  68 e1 medio es  0.2645545  y su varianza  0.002144896
##
##
## Para N:  78 e1 medio es  0.2639444  y su varianza  0.1570122
## Para N:  78 e2 medio es  0.2665661  y su varianza  0.1420078
## Para N:  78 e1 medio es  0.2635078  y su varianza  0.001810761
##
##
## Para N:  88 e1 medio es  0.2558265  y su varianza  0.1333473
## Para N:  88 e2 medio es  0.2787502  y su varianza  0.1722689
## Para N:  88 e1 medio es  0.2626849  y su varianza  0.001662892
##
##
## Para N:  98 e1 medio es  0.2530492  y su varianza  0.1311662
## Para N:  98 e2 medio es  0.2766341  y su varianza  0.1408573
## Para N:  98 e1 medio es  0.2610019  y su varianza  0.001445016
##
##
```

```
## Para N:  108 e1 medio es  0.2703801  y su varianza  0.1389497
## Para N:  108 e2 medio es  0.2545194  y su varianza  0.1173215
## Para N:  108 e1 medio es  0.257784  y su varianza  0.001212251
##
##
## Para N:  118 e1 medio es  0.2534959  y su varianza  0.1249825
## Para N:  118 e2 medio es  0.2590746  y su varianza  0.1393485
## Para N:  118 e1 medio es  0.2585452  y su varianza  0.001135226
##
##
## Los Neff para los distintos N con los que trabajamos:
##   15.29028 24.1038 36.65817 40.49829 54.02831 55.089 86.71063 80.18999 90.77146 114.6213 110.0949
## La media de los anteriores valores es:  64.36873
```