Dígitos: regresión logísitca vs. gradiente vs. vecinos

```
library(readr)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
library(tidyr)
```

Leemos los datos:

```
digitos_entrena <- read_csv('./zip-train.csv')
digitos_prueba <- read_csv('./zip-test.csv')
names(digitos_entrena)[1] <- 'digito'
names(digitos_entrena)[2:257] <- paste0('pixel_', 1:256)
names(digitos_prueba)[1] <- 'digito'
names(digitos_prueba)[2:257] <- paste0('pixel_', 1:256)
dim(digitos_entrena)
```

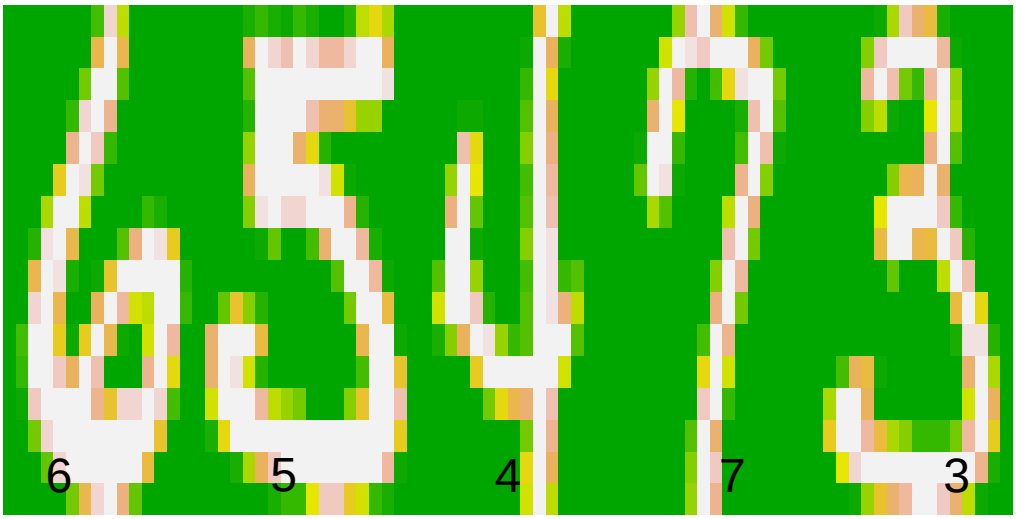
```
## [1] 7291  257
```

```
table(digitos_entrena$digito)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 1194 1005  731  658  652  556  664  645  542  644
```

Graficamos los datos:

```
graficar_digitos <- function(d_frame){
  matriz_digitos <- lapply(1:nrow(d_frame), function(i){
        matrix(as.numeric(d_frame[i, 257:2]), 16, 16)[16:1, ]
    })
    image(Reduce("rbind", matriz_digitos),
    col = terrain.colors(30), axes = FALSE)
    text(seq(0,0.9, length.out = nrow(d_frame)) + 0.05, 0.05, label = d_frame$digito, cex = 1.5)
}
graficar_digitos(digitos_entrena[1:5,])
```



Punto 0. Los datos ya están divididos en entrenamiento y prueba. Usa esta división y normaliza:

Normalizamos los datos:

```
digitos_entrena$id<-1:dim(digitos_entrena)[1]
digitos_prueba$id<-1:dim(digitos_prueba)[1]

norm<-digitos_entrena %>%
  gather(variable,valor,pixel_1:pixel_256) %>%
  group_by(variable) %>%
  summarise(media=mean(valor),de=sd(valor))

normalizar<-function(datos,norm){
  datos %>%
    gather(variable,valor,pixel_1:pixel_256) %>%
    left_join(norm) %>%
    mutate(valor_s = (valor  - media)/de) %>%
    select(id, digito, variable, valor_s) %>%
    spread(variable, valor_s)
}

digitos_entrena_norm<-normalizar(digitos_entrena,norm)
digitos_prueba_norm<-normalizar(digitos_prueba,norm)
```

En este caso resulta menos importante normalizar pues las variables tienen la misma escala de -1 a 1. No obstante, siempre es buena idea normalizarlos.

Es importante que los dígitos para los datos de prueba y de entrenamiento sean escritos por personas diferentes pues podría contaminar la predicción debido a que sería más sencillo encontrar la predicción pues los "trazos" se asemejan. Por el contrario, si lo escribe otra persona realmente revisariamos el poder predictivo de nuestro modelo.

Punto 1. Ajusta con descenso en gradiente un modelo de regresión logística, y compara con la salida de `glm` para checar tus cálculos:

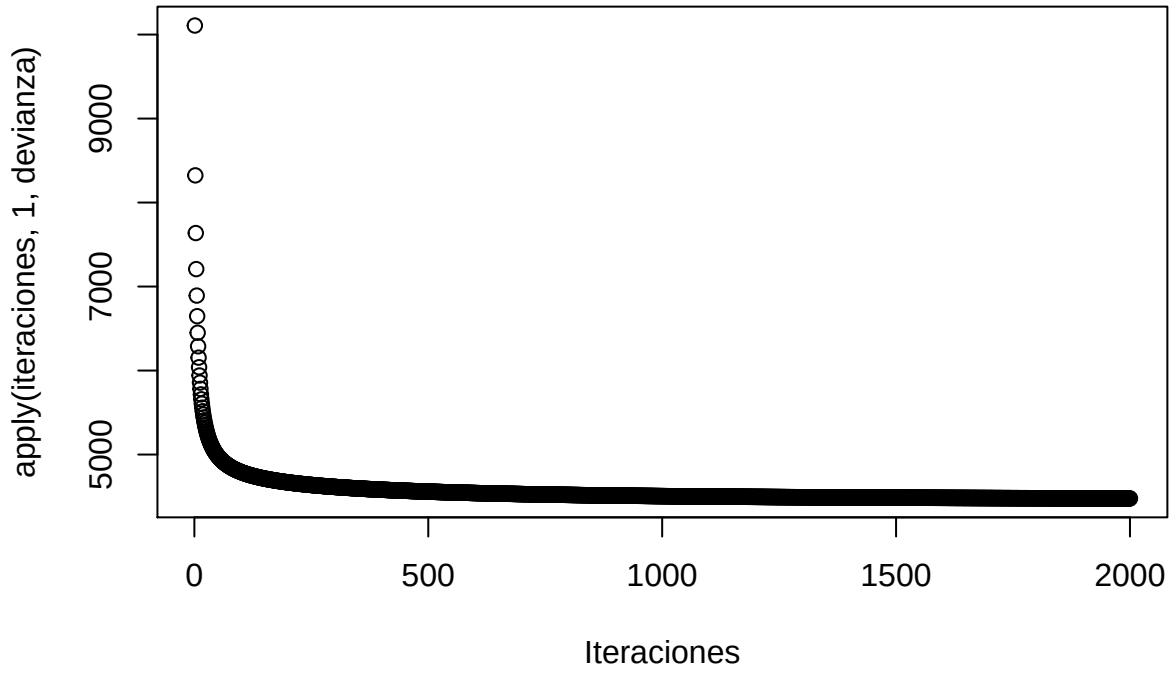
Vamos a hacer la variable dependiente y como un factor que tome 0 si es un número bajo; es decir, 0,1,2,3,4 y 1 si es un número alto; es decir, 5,6,7,8,9:

```
digitos_entrena_norm$y<-as.numeric(digitos_entrena_norm$digito>=5)
digitos_prueba_norm$y<-as.numeric(digitos_prueba_norm$digito>=5)
```

Calculamos el gradiente:

```
h <- function(z) exp(z)/(1+exp(z))

devianza_calc <- function(x, y){
  dev_fun <- function(beta){
    p_beta <- h(as.matrix(cbind(1, x)) %*% beta)
   -2*sum(y*log(p_beta) + (1-y)*log(1-p_beta))
  }
  dev_fun
}

grad_calc <- function(x_ent, y_ent){
  salida_grad <- function(beta){
```

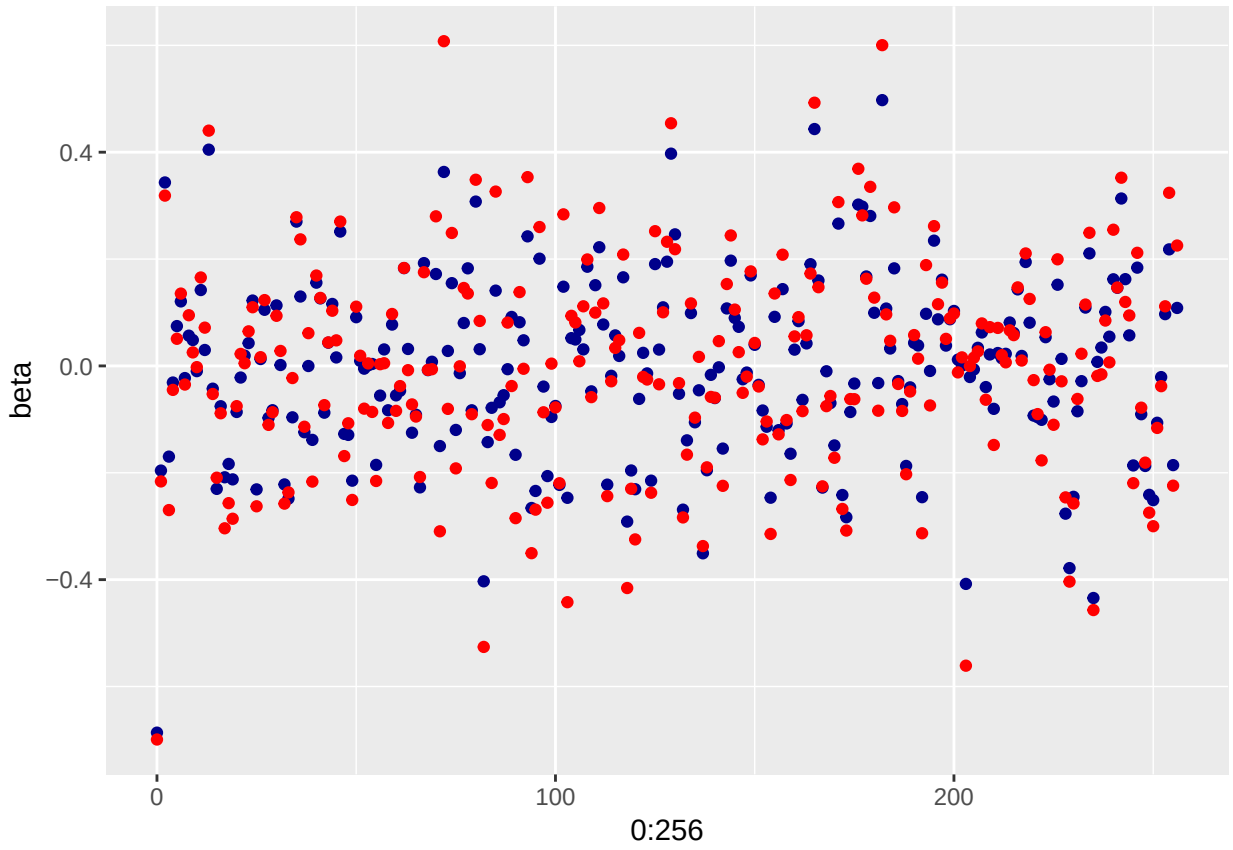
```
    p_beta <- h(as.matrix(cbind(1, x_ent)) %*% beta)
    e <- y_ent - p_beta
    grad_out <- -2*as.numeric(t(cbind(1,x_ent)) %*% e)
    names(grad_out) <- c('Intercept', colnames(x_ent))
    grad_out
  }
  salida_grad
}

descenso <- function(n, z_0, eta, h_deriv){
  z <- matrix(0,n, length(z_0))
  z[1, ] <- z_0
  for(i in 1:(n-1)){
    z[i+1, ] <- z[i, ] - eta * h_deriv(z[i, ])
  }
  z
}

devianza<-devianza_calc(as.matrix(digitos_entrena_norm[,3:258]),digitos_entrena_norm$y)
grad<-grad_calc(as.matrix(digitos_entrena_norm[,3:258]),digitos_entrena_norm$y)
iteraciones <- descenso(2000, z_0=rep(0,257), eta = 0.00001, h_deriv = grad)
plot(apply(iteraciones, 1, devianza),xlab="Iteraciones")
```




```
beta<-iteraciones[2000,]
```

Comparamos las estimaciones obtenidas utilizando descenso gradiente contra las obtenidad usando `glm`:

```
library(ggplot2)
indep<-as.matrix(digitos_entrena_norm[,3:258])
mod_log <- glm(digitos_entrena_norm$y ~ indep, family = 'binomial')
ggplot() + geom_point(aes(x=0:256, y = beta),color="darkblue") +geom_point(aes(x=0:256,y=coef(mod_log))
```



Comparamos la devianza entre descenso gradiente y `glm`:

```
mod_log$deviance
```

```
## [1] 4463.485
```

```
devianza(iteraciones[2000,])
```

```
## [1] 4477.369
```

Punto 2. Reporta devianza de entrenamiento y de prueba. Calcula también la tasa de incorrectos de prueba (usando el clasificador de máxima probabilidad):

Ahora realizaremos los calculos de la devianza promedio para la muestra de prueba:

```
dev_prueba <- devianza_calc(digitos_prueba_norm[,3:258],digitos_prueba_norm$y)
dev_prueba(iteraciones[2000,])/nrow(digitos_prueba_norm[,3:258])
```

```
## [1] 0.7932996
```

Calculamos el error de clasificación de prueba:

```
beta <- iteraciones[2000, ]
p_beta <- h(as.matrix(cbind(1, digitos_prueba_norm[,3:258])) %*% beta)
y_pred <- as.numeric(p_beta > 0.5)
mean(digitos_prueba_norm$y != y_pred)
```

```
## [1] 0.1624315
```

Punto 3. Discute por qué el enfoque de este ejercicio puede no ser tan bueno (una regresión logística para separa altos de bajos). ¿Qué otro enfoque se te ocurre usando regresión logística?:

El enfoque no es tan bueno pues la variable dependiente consta de 10 etiquetas y no 2, por lo que utilizar regresiòn logística no es del todo adecuado pues sólo nos permite utilizar 2 categorias. Se podría utilizar algun modelo multinomial donde sea posible el uso de más categorias.

Punto 4. Compara desempeño (puedes usar solo tasa de incorrectos) con algunos modelos de k-vecinos más cercanos. Por ejemplo, 1 vecino más cercano (clasificar según el ejemplo más similar):

La tasa de incorrectos utilizando $k = 1$ vecinos más cercanos:

```
library(kknn)

vmc_1 <- kknn(y~., train = digitos_entrena_norm[,3:259],test =digitos_prueba_norm[,3:259],k=1)
hat_G_1<- predict(vmc_1)
mean(digitos_prueba_norm$y != hat_G_1)
```

```
## [1] 0.04285002
```

La tasa de incorrectos utilizando $k = 5$ vecinos más cercanos:

```
vmc_5 <- kknn(y~., train = digitos_entrena_norm[,3:259],test =digitos_prueba_norm[,3:259],k=5)
hat_G_5<- predict(vmc_5)
mean(digitos_prueba_norm$y != hat_G_5)
```

```
## [1] 0.1405082
```

La tasa de incorrectos utilizando $k = 20$ vecinos más cercanos:

```
vmc_20 <- kknn(y~., train = digitos_entrena_norm[,3:259],test =digitos_prueba_norm[,3:259],k=20)
hat_G_20<- predict(vmc_20)
mean(digitos_prueba_norm$y != hat_G_20)
```

```
## [1] 0.3178874
```

Punto 5. ¿Puedes interpretar cómo funciona el modelo para hacer predicciones?:

El término $\frac{p(x)}{1-p(x)}$ corresponde a la ventaja de la respuesta $Y = 1$ para esos valores X_i. Es decir, si comparamos el cociente de las ventajas entre dos configuraciones de variables explicativas, el exponencial del parámetro asociado a la variable X_i es la cantidad por la que queda multiplicada la ventaja de respuesta $Y = 1$ cuando el valor en X_i aumenta en una unidad, sin que cambien los valores en el resto de variables explicativas.