PEC 1 - Informe: Algoritmo kNN

Jose Felix Rojas Cabeza

Contents

2020-04-04

1. El Algoritmo *k*NN

A pesar de que este algoritmo es uno de los más sencillos, es utilizado ampliamente (Lantz 2013). A continuación se indican algunas de sus fortalezas y debilidades:

Fortalezas	Debilidades
Simple y efectivo	No produce un modelo, limitando la capacidad de entender cómo se relacionan las características de los datos y la clase
No hace suposiciones sobre la distribución de datos subyacente	Requiere la selección de un k adecuado
La fase de entrenamiento es rápida	La fase de clasificación es lenta
	Las características nominales y datos faltantes requieren procesamiento adicional

El algoritmo kNN funciona utilizando información sobre los k vecinos más cercanos (Nearest Neighbors, en inglés, por eso el nombre), para clasificar ejemplos no clasificados. La letra k es un término variable, implicando que cualquier número de vecinos más cercanos podría utilizarse. Luego de escoger k, el algoritmo requiere un dataset de entrenamiento, que consista de ejemplos que se han clasificado en diferentes cateogrías, marcadas por una variable nominal. Luego, para cada registro no clasificado, kNN identifica k registros en el dataset de entrenamiento que son los más cercanos en semejanza (lo que se evalúa por la distancia, que puede ser eucilidiana, o de Malahanobis en el caso más general). Para poder comparar entre variables de manera adecuada, los datos deben escalarse.

Los pasos que se siguen en la práctica para implementar este algoritmo son: 1. Recolección de datos 2. Exploración y preparación de datos (incluye normalización) 3. Creación de datasets de entrenamiento y prueba 4. Entrenar un modelo en los datos 5. Evaluar el desempeño del modelo 6. Probar valores alternativos de k

2. Desarrollo de archivo CSV

El archivo fue generado de acuerdo a las instrucciones especificadas.

Se siguió la recomendación en la sección "De imagen a vector de las instrucciones", y se obtuvo un documento que se llamó `data.csv` para diferenciarlo del que fue proporcionado por el docente. A continuación se indican los comandos más relevantes utilizados:

```
# Ubicación del las imágenes
path_rx = "C:/Users/josef/Documents/R_Docs/M0.163/PEC1/"

# lista de imágenes
t <- list.files(path_rx, pattern= ".png")

# Caso de una sóla imagen
library(OpenImageR)

# 1. Leer la imagen
read_1 <- readImage(paste0(path_rx, t[1], sep=""))

# 2. Convertir a escala de grises, con rgb_2gray()
r2g_1 <- rgb_2gray(read_1)

# 3. Redimensionar la imagen a 64x64 pixeles con resizeImage()
redim_1 <- resizeImage(r2g_1, 64, 64, 'nearest')

# 4. Convertir la matriz a vector con as.vector()
vec_1 <- as.vector(redim_1)

# Caso 1000 imágenes

# 01. Creación de arreglo para contener los datos 512x512x3
# https://stackoverflow.com/questions/32766990/creating-a-three-dimensional-
# array-from-a-data-frame-in-r (idea obtenida de este sitio)

d <- as.data.frame( matrix( 1:(512*512*3), 512, 3) )
M <- array( unlist(d), dim=c(512, 512, 3) )

# 02. Lista de dataframes para guardar readImage[i] en bucle
lista <- list(M,...,M) # se repite el arreglo mil veces

# 03. Bucle 512x512x3
for (i in 1:1000) {lista[[i]] <- readImage(paste0(path_rx, t[i]))}

# 04. Creación de arreglo para contener los datos 512x512x1 (RGB)
d <- as.data.frame( matrix( 1:(512*512), 512, 1 ))
M2 <- array( unlist(d), dim=c(512, 512, 1) )
r2g_data <- list(M2,...,M2) # se repite el arreglo mil veces

# 05. Bucle para datos 512x512x1
for (i in 1:1000) {r2g_data[[i]] <- rgb_2gray(lista[[i]])}

# 06. Creación de arreglo para contener los datos 64x64x1
d <- as.data.frame( matrix( 1:(64*64), 1, 1 ))
M3 <- array( unlist(d), dim=c(64, 64, 1) )
```

```
# 07. Lista de dataframes para guardar resizeImage[i] en bucle
redim <- list(M3,...,M3) # se repite el arreglo mil veces

# 08. Bucle para datos 64x64
for (i in 1:1000) {redim[[i]] <- resizeImage(r2g_data[[i]], 64, 64, 'bilinear')}

# 09. Creación de vector
M4 <- c( 1:4096 )
vec <- list(M4,...,M4) # se repite el vector mil veces

# 10. Bucle para datos vectoriales
for (i in 1:1000) {vec[[i]] <- as.vector(redim[[i]])}

# 11. data frame sin "cat"
# (se hizo traspuesto para tratar que fuese más eficiente en memoria)
data_raw <- t(as.data.frame(vec))

# 12. crear cat
a <- c(rep(c("e", "n"), each=500))

# 13. agregar la columna cat
data <- cbind(a,(data_raw))

# 14. Eliminación de los nombres de fila (se hizo por conveniencia)
rownames(data) <- c(rep(c(""), each=1000))

# No se realiza normalización porque el rango de los datos está entre 0 y 1.
```

Nótese que se adjunta un archivo aparte, llamado `20200328-PEC1-parte2.Rmd`, con el cual se puede generar nuevamente el archivo de interés. la estrategia de utilizar `paste0(path_rx,t[i])` fue observada en un documento de redes neurales y deep learning(Barbero, Cardoner, and Mercader 2020).

3. Desarrollo de clasificador *k*NN

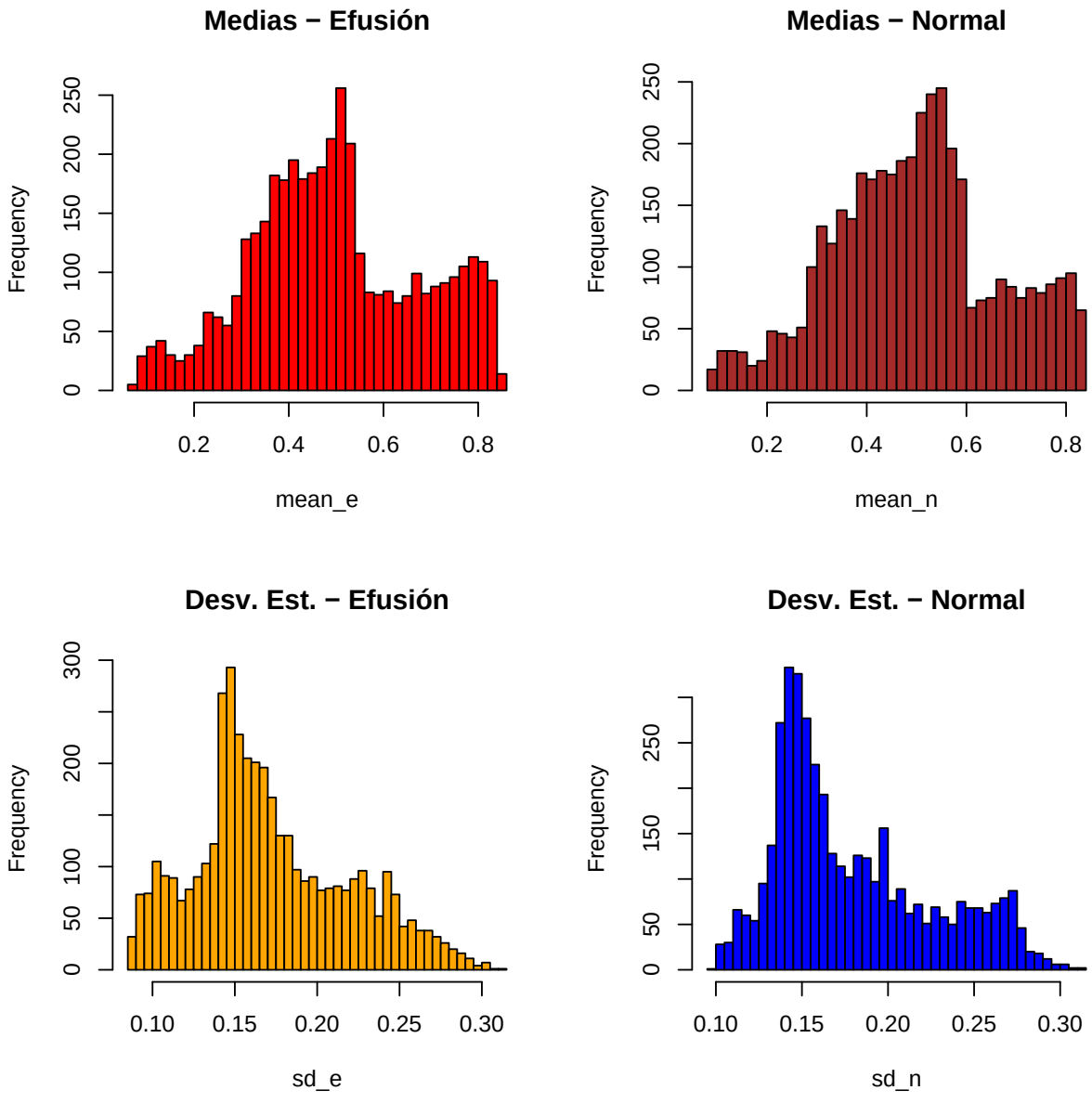
3. a) Lectura de `Rx_Torax_4097.csv`

```
> Rx_t <- read.csv(params$data, stringsAsFactors = FALSE)
>
> # eliminación de la columna de nombres (vaciada por conveniencia) y
> # guardado en el mismo data set
> Rx_t <- Rx_t[-1]
>
> # Verificación de la estructura (7/4096 variables)
> head(Rx_t[,1:7], 5)
```

```
  cat         v2         v3         v4         v5         v6         v7
1   e 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
2   e 0.00000000 0.16078431 0.27843137 0.33333333 0.36470588 0.38039216
3   e 0.84313725 0.81568627 0.19607843 0.14117647 0.17647059 0.17254902
4   e 0.57647059 0.58823529 0.59215686 0.59215686 0.59607843 0.58039216
5   e 0.03921569 0.04313725 0.03921569 0.03921569 0.03921569 0.03529412
```

3. b) Descripción gráfica de los datos

```
> par(mfrow = c(2,2))
> hist(mean_e, breaks = 52, main = "Medias - Efusión", col="red")
> hist(mean_n, breaks = 52, main = "Medias - Normal", col="brown")
> hist(sd_e, breaks = 70, main = "Desv. Est. - Efusión", col="orange")
> hist(sd_n, breaks = 70, main = "Desv. Est. - Normal", col="blue")
```



Se observa que las distribuciones de medias se comportan de manera similar en los extremos. Sin embargo, en el intervalo de 0.4 a 0.6 se notan diferencias.

Las desviaciones estándar tienen picos en 0.15, con distribuciones similares; pero la desviación en el caso de efusión tiene una amplitud algo mayor (nótese que en el gráfico amarillo se observa un pequeño pico cerca de 0.10). En el caso normal se ve una desviación más irregular, lo que se hace más obvio comparando el

intervalo entre 0.20 y 0.30.

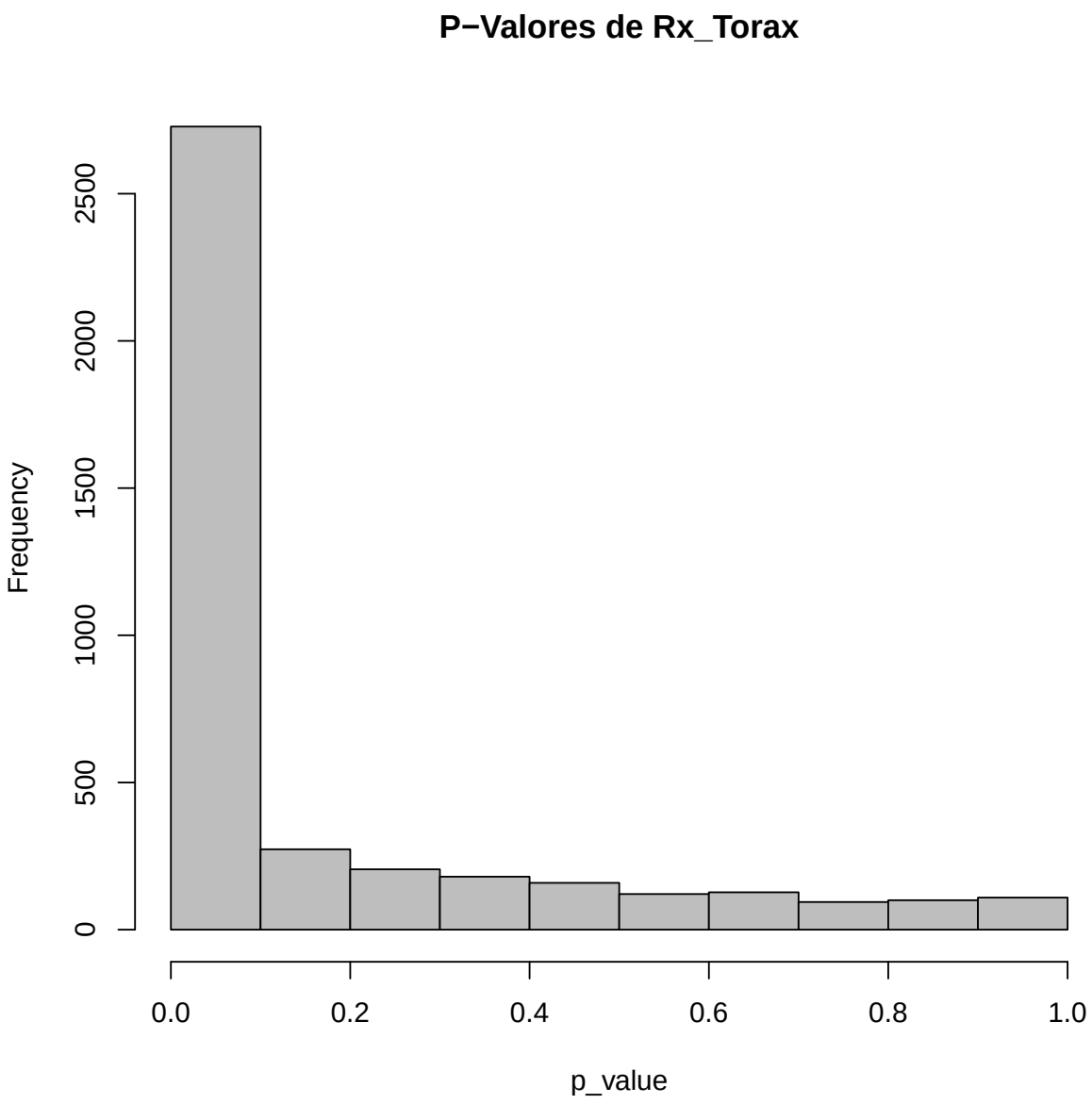
Mi interpretación de los resultados es que la forma como estamos midiendo normal y efusión deja semejanzas en las columas cerca de los extremos de la imagen, lo que explicaría la semejanza que se observa en los extremos de los gráficos de las medias. Interesantemente, esa semejanza se traslada bien a los histogramas de dispersión, y se nota que se pueden hacer distinciones interesantes, porque la variabilidad entre los grupos es notable.

3. c) Contraste de valores:

```
> # Caso de un sólo Valor
> mean_comparison_1 <- list()
>
> mean_comparison_1 <- t.test(Rx_md_e[1:500,1], Rx_md_n[1:500,1],
+                             alternative = c("two.sided"), var.equal = F)
>
> # Caso de 4096 valores
> # Se comparan las 4096 variables, con H0: mean(e) = mean(n) para cada variable.
> mean_comparison <- list()
>
> for(i in 1:4096) {
+  mean_comparison[[i]] <- t.test(Rx_md_e[1:500,i], Rx_md_n[1:500,i],
+                                 alternative = c("two.sided"), var.equal = F)
+ }
```

```
> p_value <- c()
>
> for(i in 1:4096){
+   p_value[i] <- mean_comparison[[i]]$p.value
+ }
```

```
> # Histograma
> hist(p_value, col="grey", main = "P-Valores de Rx_Torax")
```



La distribución de un histograma de los p-valores nos indica que es posible encontrar diferencias entre los grupos.

Procedimiento de Benjamini-Hochberg:

La primera aproximación para controlar la tasa de hallazgos falsos (False Discovery Rate, FDR) fue descrita por los autores anteriormente mencionados (Benjamini and Hochberg 1995). Si se desea controlar que en un estudio con n comparaciones el FDR no supere un porcentaje α hay que:

1. Para un dado α, encuentre el $max(k)$ tal que $P(k) \leq \frac{k}{m} \cdot \alpha$.
2. Rechazar la hipótesis nula (i. e. declarar descubrimientos) para todos los $H(i)$ para $i = 1, \cdots, k$.

Geométricamente, esto corresponde a graficar el P(k) vs k (en los ejes x e y respectivamente), trazando la línea a través del origen con una pendiente $\frac{\alpha}{m}$, y declarando los descubrimientos para todos los puntos en la izquierda, hasta el último punto bajo la línea.

El procedimiento BH es válido cuando los m tests son independientes y también en varios escenarios de dependencia, pero no es universalmente válido. También satisface la desigualdad $E(Q) \leq \frac{m_0}{m} \cdot \alpha \leq \alpha$.

```
> adjusted_pval <- p.adjust(p_value,method="BH")
> # http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R-Manual/R-Manual22.html
```

```
> c3 <- sort(adjusted_pval)
>
> c3_2 <- head(c3, 26)
```

```
> c2 <- list()
>
> for (i in 1:length(c3)) {
+ c2[[i]] <- which(adjusted_pval == c3[i])
+ }
>
> head(c2, 26)
```

```
[[1]]
[1] 2987

[[2]]
[1] 2859

[[3]]
[1] 2986

[[4]]
[1] 2922 3051

[[5]]
[1] 2922 3051

[[6]]
[1] 2923

[[7]]
[1] 1321 3052

[[8]]
[1] 1321 3052

[[9]]
[1] 1384 1450 2924

[[10]]
[1] 1384 1450 2924

[[11]]
[1] 1384 1450 2924

[[12]]
[1] 2985

[[13]]
```

```
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[14]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[15]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[16]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[17]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[18]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[19]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[20]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[21]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[22]]
 [1] 1000 1063 1064 1127 1190 1255 1449 2860 2921 2988

[[23]]
[1] 1191

[[24]]
[1] 1447

[[25]]
[1] 2920 3050

[[26]]
[1] 2920 3050
```

```
> # Lo he hecho a mano porque no se me ocurrió otra manera
> c2_2 <- c(2987, 2859, 2986, 2922, 3051, 2923, 1321, 3052, 1384, 1450,
+           2924, 2985, 1000, 1063, 1064, 1127, 1190, 1255, 1449, 2860,
+           2921, 2988, 1191, 1447, 2920, 3050)
```

```
> c1 <- c()
>
> for(i in 1:length(c2_2)){
+   c1[i] <- mean_e[as.integer(c2_2[i])] - mean_n[as.integer(c2_2[i])]
+ }
```

```
> library(knitr, quietly = TRUE)
> datos <- cbind(c2_2, c1, c3_2)
>
```

```
> kable(datos, col.names = c("Número", "Dif. Medias", "P-Valor"),
+       align = c("c", "c", "c"), caption = "Contraste de Valores Medios")
```

Table 2: Contraste de Valores Medios

Número	Dif. Medias	P-Valor
2987	0.0122745	0
2859	0.0111216	0
2986	0.0172078	0
2922	0.0189176	0
3051	0.0287216	0
2923	0.0438980	0
1321	0.0579451	0
3052	0.0688706	0
1384	0.0816078	0
1450	0.0893882	0
2924	0.0942902	0
2985	0.0985020	0
1000	0.0975608	0
1063	0.0988471	0
1064	0.1026588	0
1127	0.1026196	0
1190	0.1079529	0
1255	0.1018588	0
1449	0.1012078	0
2860	0.1006745	0
2921	0.0949255	0
2988	0.0963922	0
1191	0.0973647	0
1447	0.0965647	0
2920	0.0956784	0
3050	0.0968627	0

3. d) Implementación del algoritmo *k*NN

3. d) 1. Datasets de entrenamiento y prueba:

Utilizando la semilla aleatoria 123, separar los datos en dos partes, una parte para training (67 %) y una parte para test (33 %).

```
> # Semilla
> set.seed(123)
>
> # Train / Test (.67/.33)
> p <- .67
>
> n_train <- round(500*.67, 0) # 335
> n_test <- round(500*.33, 0)  # 165
```

```
> # Todos los datos
> all <- data.frame()
> for(i in 1:1000){
+   all[i,1:4097] <- Rx_t[i,1:4097]
+ }
```

```
> # efusion 1:335
> train_1 <- data.frame()
> for(i in 1:335){
+   train_1[i,1:4097] <- Rx_t[i,1:4097]
+ }
```

```
> # normal 501:836
> train_2 <- data.frame()
> for(i in 1:335){
+   train_2[i+500,1:4097] <- Rx_t[i+500,1:4097]
+ }
```

```
> train_2 <- narm(train_2)
>
> head(train_2[,1:6])
```

```
    cat         v2          v3          v4          v5          v6
501   n 0.03137255 0.031372549 0.027450980 0.023529412 0.019607843
502   n 0.00000000 0.105882353 0.098039216 0.094117647 0.094117647
503   n 0.00000000 0.058823529 0.537254902 0.560784314 0.607843137
504   n 0.00000000 0.000000000 0.000000000 0.000000000 0.000000000
505   n 0.07843137 0.074509804 0.074509804 0.070588235 0.050980392
506   n 0.01568627 0.007843137 0.007843137 0.007843137 0.007843137
```

```
> tail(train_2[,1:6])
```

```
    cat          v2          v3         v4         v5         v6
830   n 0.000000000 0.000000000 0.00000000 0.00000000 0.00000000
831   n 0.000000000 0.047058824 0.03921569 0.03529412 0.16862745
832   n 0.000000000 0.000000000 0.00000000 0.00000000 0.00000000
833   n 0.113725490 0.003921569 0.03921569 0.06274510 0.05882353
834   n 0.003921569 0.003921569 0.00000000 0.00000000 0.00000000
835   n 0.000000000 0.000000000 0.00000000 0.00000000 0.00000000
```

```
> # efusion 336:500
> test_1 <- data.frame()
> for(i in (n_train+1):500){
+   test_1[i,1:4097] <- Rx_t[i,1:4097]
+ }
```

```
> test_1 <- narm(test_1)
>
> head(test_1[,1:6], 5)
```

```
    cat         v2         v3         v4         v5         v6
336   e 0.01960784 0.01960784 0.01960784 0.01960784 0.01960784
337   e 0.01176471 0.01176471 0.01176471 0.01176471 0.01176471
338   e 0.01960784 0.01568627 0.01568627 0.01568627 0.01568627
339   e 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
340   e 0.05882353 0.05490196 0.05098039 0.04705882 0.04313725
```

```
> tail(test_1[,1:6], 5)
```

```
    cat         v2          v3        v4         v5        v6
496   e 0.14117647 0.137254902 0.1294118 0.12549020 0.1215686
497   e 0.00000000 0.000000000 0.0000000 0.05882353 0.1568627
498   e 0.01176471 0.003921569 0.0000000 0.00000000 0.0000000
```

```
499   e 0.12156863 0.062745098 0.0745098 0.29803922 0.5607843
500   e 0.00000000 0.000000000 0.0000000 0.00000000 0.0000000
```

```
> # normal 836:1000
> test_2 <- data.frame()
> for(i in (501+n_train):1000){
+   test_2[i,1:4097] <- Rx_t[i,1:4097]
+ }
```

```
> test_2 <- narm(test_2)
>
> head(test_2[,1:6], 5)
```

```
    cat         v2         v3          v4         v5         v6
836   n 0.03921569 0.03921569 0.168627451 0.27843137 0.38823529
837   n 0.00000000 0.00000000 0.000000000 0.00000000 0.00000000
838   n 0.10588235 0.09803922 0.090196078 0.08235294 0.07058824
839   n 0.00000000 0.00000000 0.003921569 0.01568627 0.01568627
840   n 0.16862745 0.25098039 0.341176471 0.38431373 0.37647059
```

```
> tail(test_2[,1:6], 5)
```

```
     cat         v2         v3          v4         v5         v6
996    n 0.02745098 0.02745098 0.027450980 0.02745098 0.02745098
997    n 0.00000000 0.00000000 0.007843137 0.01176471 0.01176471
998    n 0.47843137 0.00000000 0.019607843 0.05882353 0.06274510
999    n 0.04705882 0.04705882 0.047058824 0.04705882 0.04705882
1000   n 0.53725490 0.54117647 0.600000000 0.63921569 0.72941176
```

```
> # Comprobación de diumensiones
> dim(train_1)
```

```
[1]  335 4097
```

```
> dim(train_2)
```

```
[1]  335 4097
```

```
> dim(test_1)
```

```
[1]  165 4097
```

```
> dim(test_2)
```

```
[1]  165 4097
```

```
> train_0 <- rbind(train_1, train_2)
>
> test_0 <- rbind(test_1, test_2)
>
> dim(train_0)
```

```
[1]  670 4097
```

```
> dim(test_0)
```

```
[1]  330 4097
```

```
> set.seed(123)
> # Reordena las posiciones para que las muestras queden al azar
>
```

```
> shuffle_train <- c(sample(1:670))
> shuffle_test <- c(sample(1:330))
```

```
> length(shuffle_test)
```

```
[1] 330
```

```
> length(shuffle_train)
```

```
[1] 670
```

```
> train_shuffled <- data.frame()
> for(i in 1:length(shuffle_train)){
+   train_shuffled[i,1:4097] <- train_0[as.integer(shuffle_train[i]), 1:4097]
+ }
```

```
> test_shuffled <- data.frame()
> for(i in 1:length(shuffle_test)){
+   test_shuffled[i,1:4097] <- test_0[as.integer(shuffle_test[i]), 1:4097]
+ }
```

```
> train_labels <-c()
> for(i in 1:length(shuffle_train)){
+ train_labels[i] <- train_shuffled[i , 1]
+ }
```

```
> test_labels <-c()
> for(i in 1:length(shuffle_test)){
+ test_labels[i] <- test_shuffled[i , 1]
+ }
```

```
> dim(train_shuffled)
```

```
[1]  670 4097
```

```
> dim(test_shuffled)
```

```
[1]  330 4097
```

```
> table(train_labels)
```

```
train_labels
  e   n
335 335
```

```
> table(test_labels)
```

```
test_labels
  e   n
165 165
```

```
> train <- train_shuffled[ , 2:4097]
> test <- test_shuffled[ , 2:4097]
>
>
> dim(train)
```

```
[1]  670 4096
```

```
> dim(test)
```

```
[1]  330 4096
```

3. d) 2. Predicción de derrame

Utilizar un kNN ($k = 3, 5, 7, 11, 23, 45, 67$) basado en el training para predecir que radiografia es normal o con derrame.

```
> library(class)
> pred_03 <- knn(train = train, test = test, cl = train_labels , k = 3)
> pred_05 <- knn(train = train, test = test, cl = train_labels , k = 5)
> pred_07 <- knn(train = train, test = test, cl = train_labels , k = 7)
> pred_11 <- knn(train = train, test = test, cl = train_labels , k = 11)
> pred_23 <- knn(train = train, test = test, cl = train_labels , k = 23)
> pred_45 <- knn(train = train, test = test, cl = train_labels , k = 45)
> pred_67 <- knn(train = train, test = test, cl = train_labels , k = 67)
```

```
> # se muesta sólo $t para ahorrar espacio
> p_03$t # 0.318 true positive
```

```
   y
x     e   n
  e 105  60
  n  71  94
```

```
> p_05$t # 0.309 true positive
```

```
   y
x     e   n
  e 102  63
  n  68  97
```

```
> p_07$t # 0.321 true positive
```

```
   y
x     e   n
  e 106  59
  n  72  93
```

```
> p_11$t # 0.339 true positive
```

```
   y
x     e   n
  e 112  53
  n  66  99
```

```
> p_23$t # 0.342 true positive
```

```
   y
x     e   n
  e 113  52
  n  58 107
```

```
> p_45$t # 0.361 true positive
```

```
   y
x     e   n
  e 119  46
```

```
  n  54 111
```

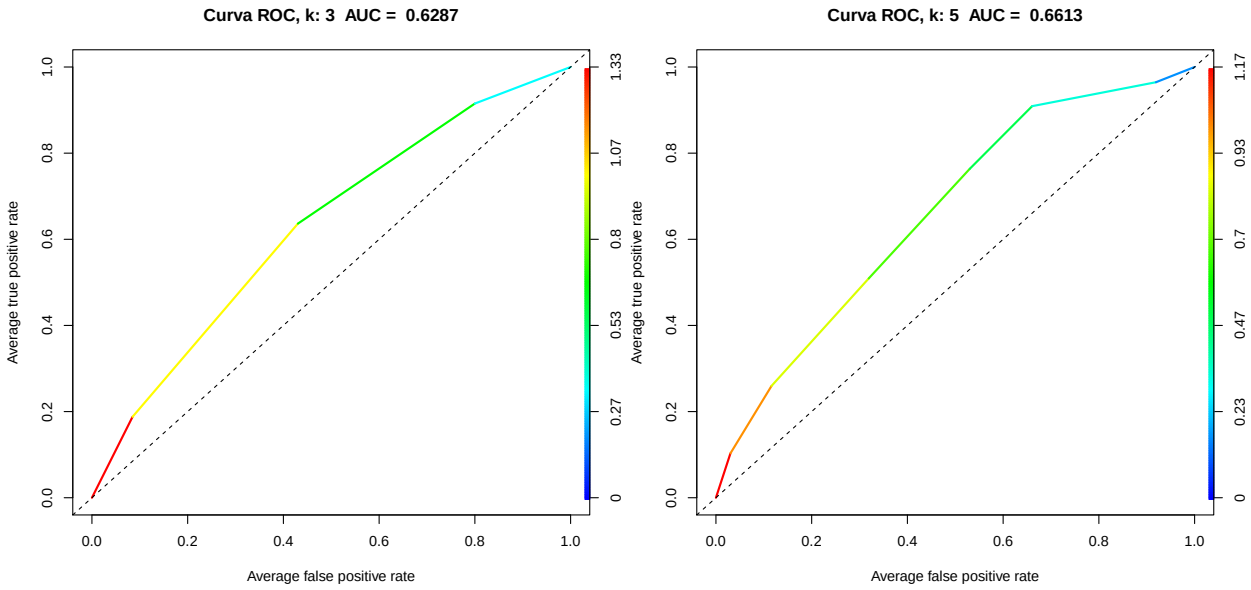
```
> p_67$t # 0.372 true positive
```

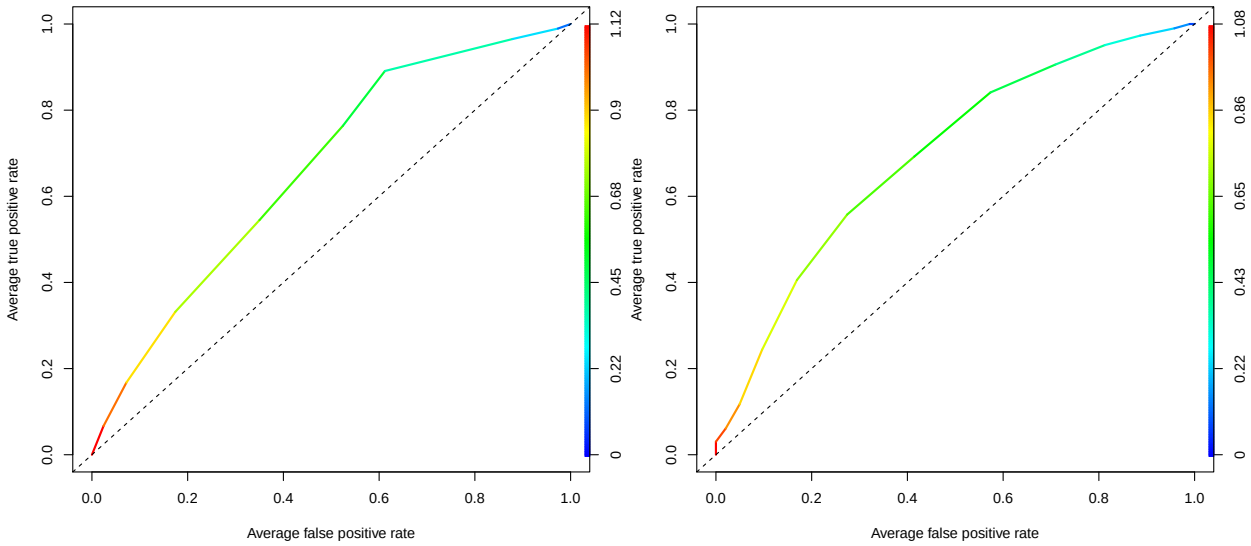
```
   y
x     e   n
  e 123  42
  n  56 109
```

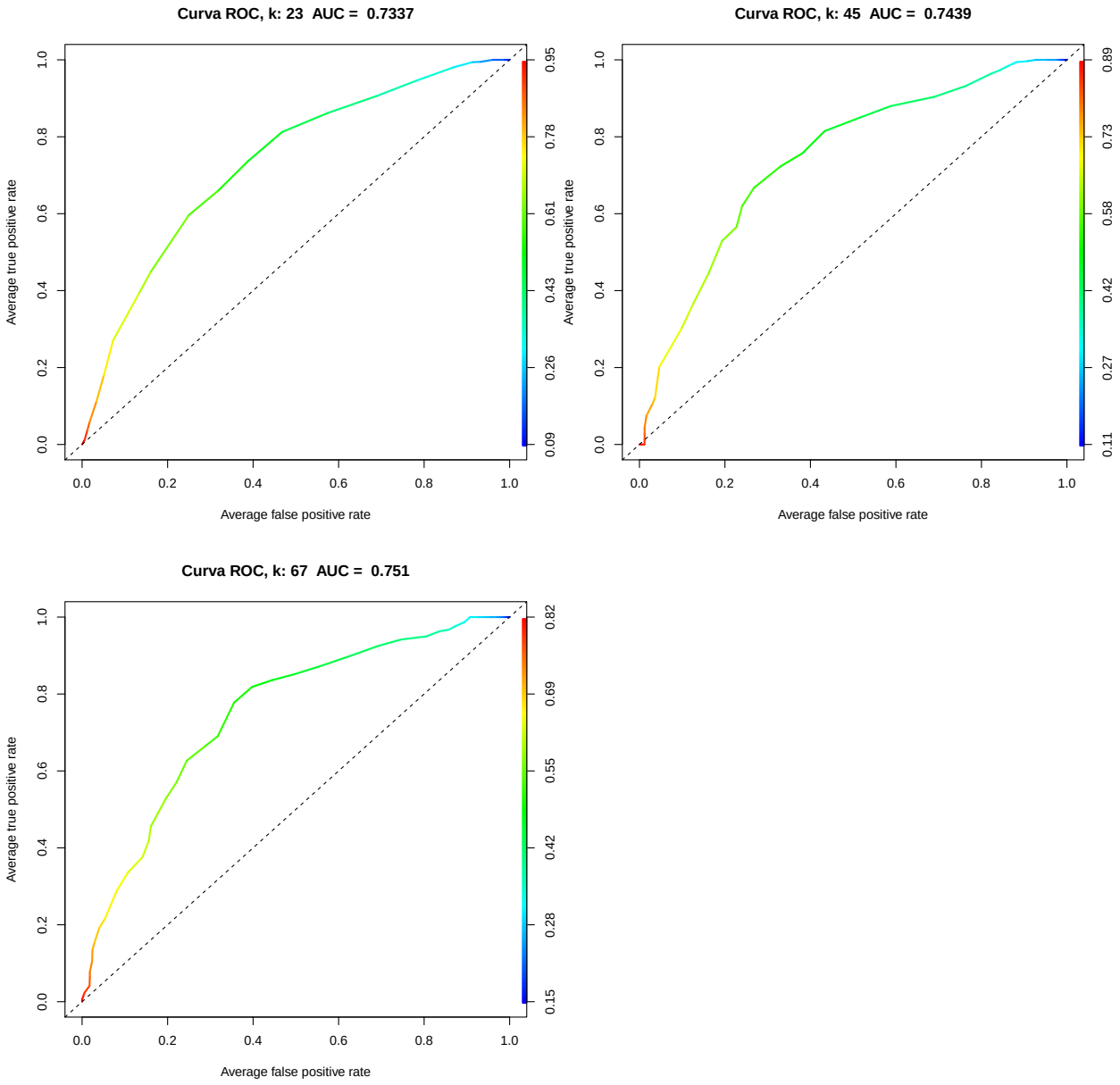
3. d) 3. Curvas ROC

Para cada k realizar una curva ROC que muestre además el AUC.

```
> # caso varias curvas
> library(ROCR)
> library(pROC)
> ks <- c( 3, 5, 7, 11, 23, 45, 67)
>
> par(mfrow=c(2,2))
>
> for(i in ks){
+   test_pred <- knn(train=train, test=test, cl=train_labels, k= i, prob = T)
+   prob <- attr(test_pred, "prob")
+
+   prob_tf <- ifelse(test_pred=="e", prob, 1-prob)
+
+   area <- auc(test_labels, prob_tf)
+
+   p_knn <- prediction(prob_tf, test_labels,
+                       label.ordering = c("n", "e"))
+   p_knn <- performance(p_knn, "tpr", "fpr")
+
+   plot(p_knn, avg="threshold", colorize=T, lwd=2,
+        main= paste("Curva ROC, k:", i, " AUC = ",
+                    round(area,4)))
+   abline(a=0, b=1, lwd=1, lty=2)
+ }
```


Curva ROC, k: 3 AUC = 0.6287
Curva ROC, k: 5 AUC = 0.6613
Average true positive rate
Average false positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0
0.27
0.53
0.8
1.07
1.33
0.23
0.47
0.7
0.93
1.17


Curva ROC, k: 7 AUC = 0.6621
Curva ROC, k: 11 AUC = 0.6954
Average true positive rate
Average false positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0
0.22
0.45
0.68
0.9
1.12
0.43
0.65
0.86
1.08



3. d) 4. Evaluación de la clasificación

Comentar los resultados de la clasificación en función de la curva ROC y del número de falsos positivos, falsos negativos y error de clasificación obtenidos para los diferentes valores de k. La clase que será asignada como positiva es la e.

La curva ROC indica que la clasificación con los distintos algoritmos kNN es mejor que hacer la clasificación al azar, pero el rendimiento de los primeros 4 algortimos fue pobre, mientras que el de los últimos 3 fue aceptable.

Valor k	Falsos negativos	Falsos positivos	Porcentaje clasificado incorrectamente
3	71	60	0.3969697
5	68	63	0.3969697
7	72	59	0.3969697

Valor k	Falsos negativos	Falsos positivos	Porcentaje clasificado incorrectamente
11	66	53	0.3606061
23	58	52	0.3333333
45	54	46	0.3030303
67	56	42	0.2969697

Cuando se trata de diagnósticos, los casos de falsos negativos pueden ser extremadamente costosos. En este caso, una persona con efusión pleural creería que no tiene inconvenientes, lo que podría resultar en un problema posteriormente. Los falsos positivos dan una "falsa alarma" a quien recibe el diagnóstico. Si bien es cierto que son menos peligrosos que un falso negativo, lo ideal sería evitarlos, para no generar carga financiera para el sistema médico, o estrés para el paciente, debido a más pruebas o tratamientos.(Lantz 2013)

Referencias

Barbero, Mattia, David Cardoner, and Arnau Mercader. 2020. "Artificial Neural Networks & Deep Learning." https://agorastats.github.io/Notes/report_nets.html.

Benjamini, Yoav, and Yosef Hochberg. 1995. "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing." *Journal of the Royal Statistical Society: Series B (Methodological)* 57 (1): 289–300.

Lantz, Brett. 2013. *Machine Learning with R.* Packt Publishing Ltd.