Teoría de Detección y Estimación - 86.55
Trabajo Final - Estimación y clasificación no paramétrica

Sampayo, Sebastián Lucas
E-Mail: sebisampayo@gmail.com
Padrón: 93793

Primer Cuatrimestre de 2014
14 de Julio

Índice

1. a) Para dos distribuciones y probabilidades a priori dadas, genere $N1 = N2 = 10^4$ muestras de cada una.

$F1 = Uniforme[0, 10]$ *y* $F2 = Gaussiana(2, 1)$

En este caso se utilizaron lás clásicas funciones '*rand()*' y '*normrnd()*' con los datos pedidos. El código es básico y se puede ver en el archivo fuente (archivo: "TrabajoFinal.m").

2. b) Estime las diferentes $F_X(x)$ utilizando *'Parzen windows'* con ventanas según se pide.

Ventana de Parzen: Uniforme de longitud 'h' (elija un valor de 'h' y justifique su elección).

Este método se basa en sumar ventanas de longitud fija, centradas en las muestras, cuya altura depende de la cantidad de muestras que se encuentren dentro del dominio de esta:

$$\boxed{\hat{p}(x) = \frac{1}{Nh} \sum_{i=1}^{N} \varphi\left(\frac{x - x_i}{h}\right)}$$

donde,
$\hat{p}(x)$: Densidad de probabilidad estimada.
x_i: N muestras de entrenamiento
φ : Ventana
h : Longitud de la ventana

Para este ejercicio se codificó una rutina especial (archivo: "parzen_estimate.m") para calcular la densidad de probabilidad estimada (por ventanas de Parzen) dado un vector de muestras de entrenamiento, un valor para la longitud de la ventana y el tamaño aproximado del espacio creado para la densidad de probabilidad (tamaño del vector *'p'*). La fórmula teórica se llevó a cabo computacionalmente con la operación de convolución "*conv()*" de MATLAB.

```
% --- Función estimadora por ventanas de Parzen ---
%
% [x, p] = parzen_estimate(data, h, n)
%
% p: función de densidad de probabilidad estimada
% x: espacio creado donde se evalúa 'p'. (p = p(x))
% data: muestras de entrenamiento
% h: longitud de la ventana de Parzen
% n: cantidad de muestras del espacio creado para 'p' (length(p)~n)
%
% Ejemplo:
% data = rand(1e4,1);
% [x, p] = parzen_estimate(data, 0.1, 1e5);
% plot(x,p)
```

A la hora de elegir la longitud de la ventana de Parzen, se pensó en $\hat{p}(x)$ como una función de parámetro *'h'*, esto es $\hat{p}(x, h)$. De esta manera se puede avanzar de un problema de estimación no-paramétrica a uno de estimación paramétrica, cuyo parámetro a estimar es *'h'*. De acuerdo a esto se tomó el siguiente criterio: dividir las muestras en 2 y hacer "*maximum likelihood*". Con una de estas porciones de los datos se realizó la estimación para un rango de valores de *'h'*. Luego con la otra porción se calculó el *"likelihood"* y se buscó el máximo de esta función. El valor inicial de 'h' se tomó arbitrariamente cercano a:

$$h_{inicial} = \frac{max(data) - min(data)}{\sqrt{N}}$$

para luego ir ajustando el rango.

A modo de ejemplo ilustrativo, se puede ver la siguiente imagen donde se graficaron varias densidades según un h distinto (solo se muestran algunas curvas para no perder claridad). En línea punteada se observa la densidad de probabilidad real.

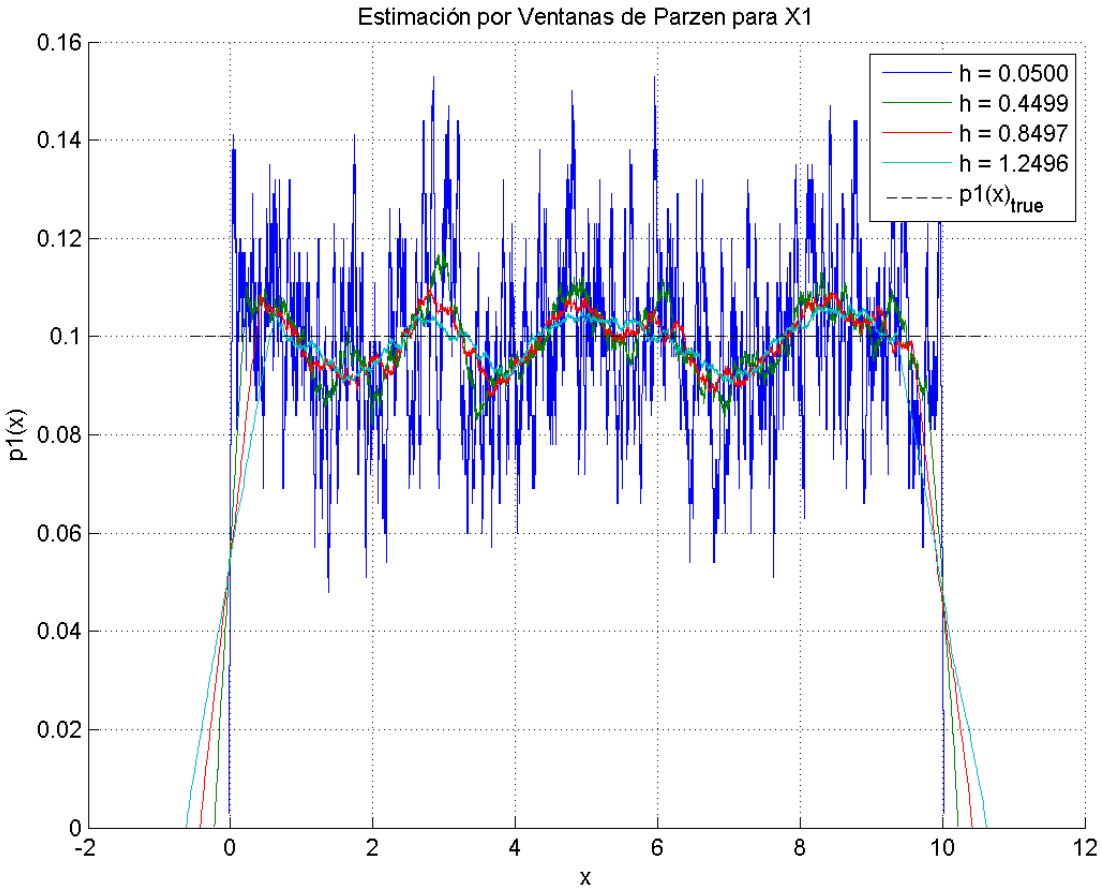

Figura 1: Estimación para distintos valores de 'h', caso X1.

Luego, el *likelihood* quedó de la siguiente manera:

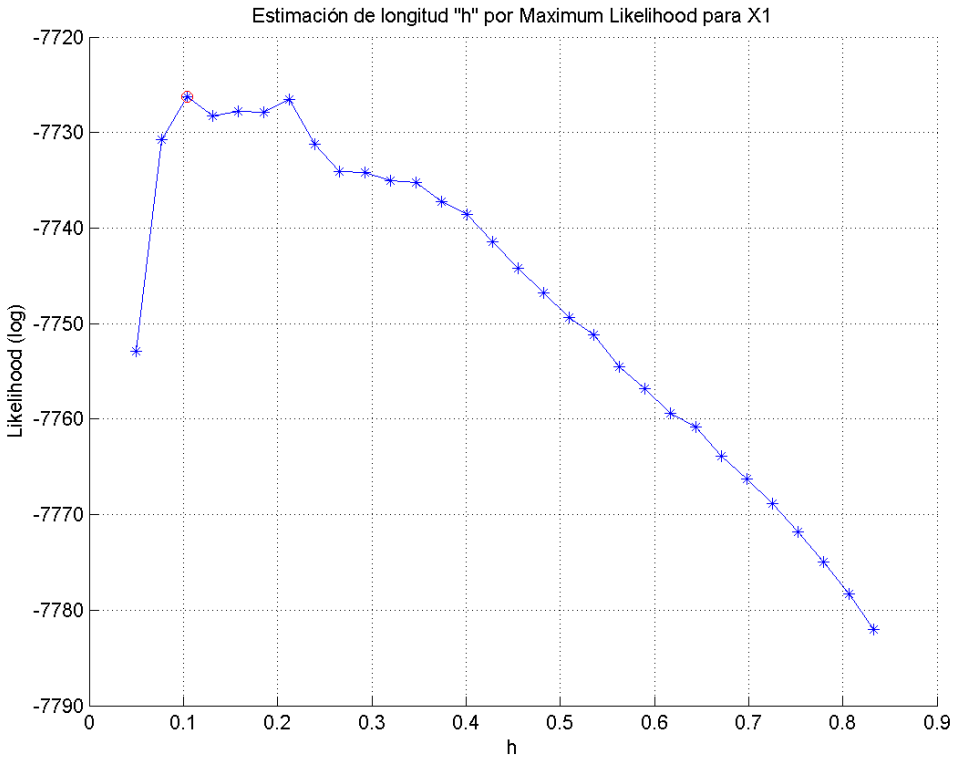

Figura 2: *Likelihood*, caso X1.

En el caso de la clase 2, se obtuvieron los siguientes resultados con el mismo procedimiento:

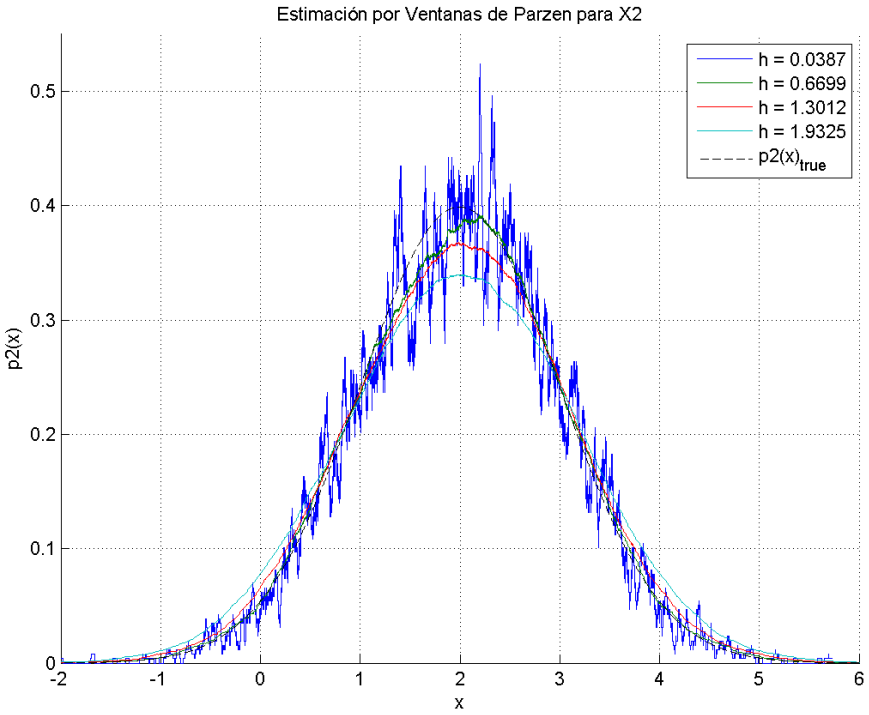

Figura 3: Estimación para distintos valores de 'h', caso X2.

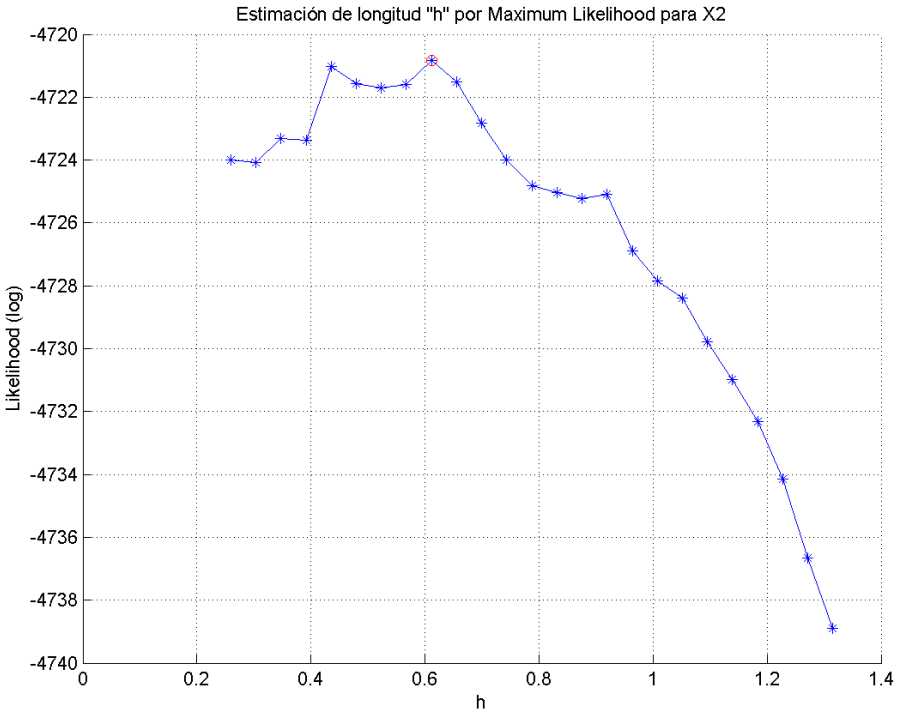

Figura 4: *Likelihood*, caso X2.

Una vez obtenido el h "óptimo" según el criterio de *Máximum Likelihood,* se estimó nuevamente la densidad de probabilidad de la muestra. En la siguiente figura se muestran los resultados de ambas clases, pesadas por sus respectivas probabilidades *a priori*.

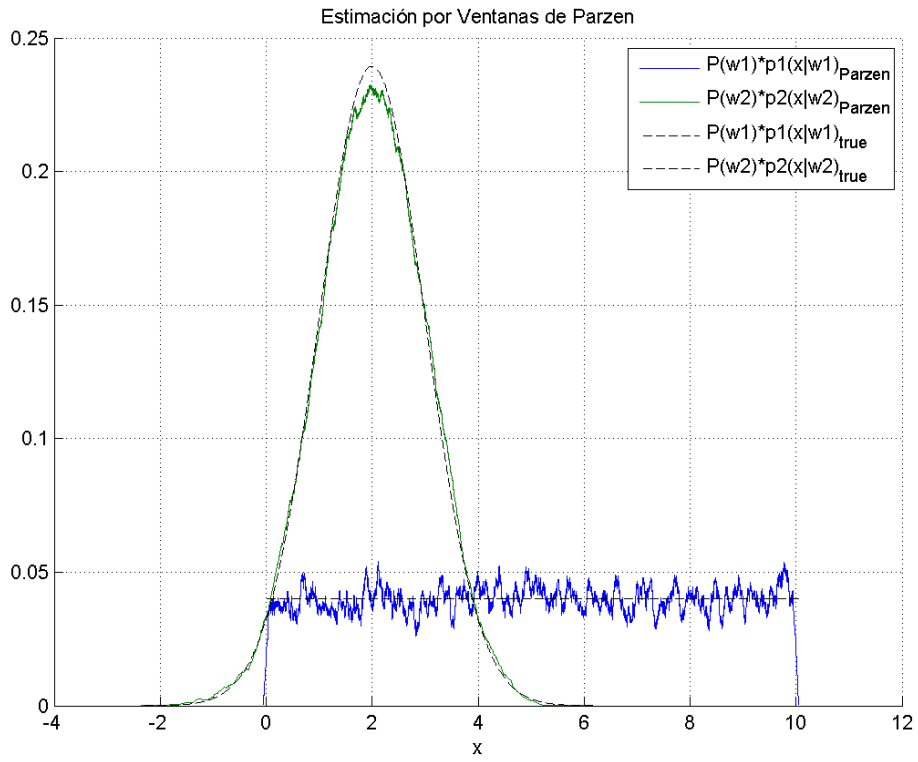

Figura 5: Resultado de las estimación con "h óptimo".

3. c) Estime las diferentes $F_X(x)$ utilizando Kn vecinos más cercanos para diferentes valores de $k = 1, 10, 50, 100$.

Salvando el problema de la elección del ancho de la ventana y de la perdida de resolución en Parzen, se tiene el método de los K vecinos basado en:

$$\boxed{\hat{p}(x) = \frac{k/N}{V(x)}}$$

donde $V(x)$ es el volumen de la región que contiene a los 'k' vecinos y a x.

Análogamente al ejercicio anterior, en este apartado se codificó una rutina especial (archivo: "knn_estimate.m") para calcular la densidad de probabilidad estimada (por Kn vecinos más cercanos) dado un vector de muestras de entrenamiento, un valor de 'K' y el tamaño aproximado del espacio creado para la densidad de probabilidad (tamaño del vector *'p'*). Este algoritmo está basado en la función "*knnsearch()*" de MATLAB.

```
% --- Función estimadora por Kn vecinos más cercanos ---%
% [x, p] = knn_estimate(data, k, n)
%
% p: función de densidad de probabilidad estimada
% x: espacio creado donde se evalúa 'p'. (p = p(x))
% data: muestras de entrenamiento
% k: cantidad de vecinos más cercanos
% n: cantidad de muestras del espacio creado para 'p' (length(p)~n)
%
% Ejemplo:
% data = rand(1e4,1);
% [x, p] = knn_estimate(data, 3, 1e5);
% plot(x,p)
```

Con este método de estimación, se pudo observar un nivel de "ruido" muy grande para la mayoría de los casos de 'k'. Sin embargo, se notó cómo la curva se suaviza a medida que la cantidad de vecinos cercanos que se toma ('k') aumenta.

En las siguientes figuras se observan los resultados para k = 10 y k = 100. El gráfico cuando k = 1 se torna bastante caótico, y el caso k = 50, es extrapolable en base a las curvas presentadas.

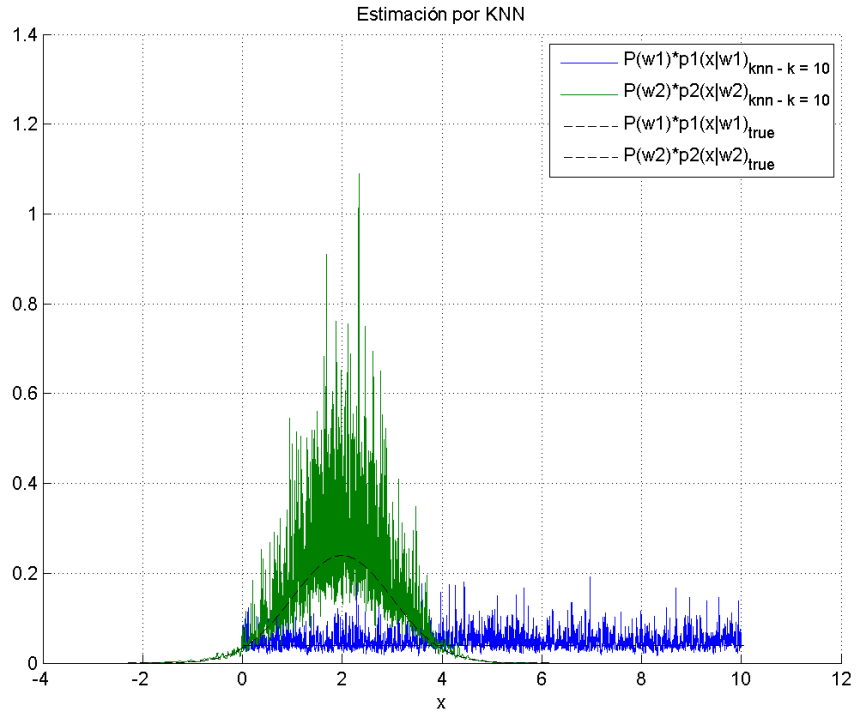

Figura 6: Estimación por KNN, k = 10.

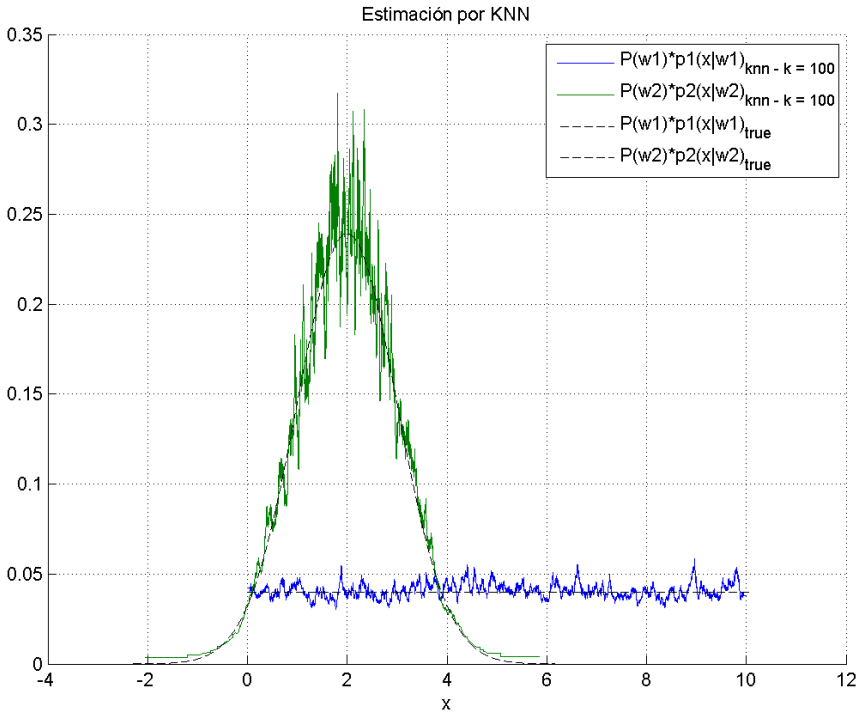

Figura 7: Estimación por KNN, k = 100.

4. d) Para b) y c) realice un clasificador y clasifique 10^2 nuevas muestras, mida el error obtenido.

A la hora de realizar un clasificador, se utilizó la regla de máxima probabilidad *a posteriori*. Esto es, dada una observación x, se decide según:

$$p_0(w_0|x) \underset{\alpha_1}{\overset{\alpha_0}{\gtrless}} p_1(w_1|x)$$

Utilizando Bayes, la regla queda como sigue:

$$\boxed{p_0(x|w_0)P(w_0) \underset{\alpha_1}{\overset{\alpha_0}{\gtrless}} p_1(x|w_1)P(w_1)}$$

En concreto, en esta sección se realizó una función (archivo: "bayesian_classificate.m") cuyo propósito es clasificar una muestra en base a 2 clases, dadas sus densidades de probabilidad y probabilidades *a priori* (clasificador bayesiano).

```
% --- Función de Clasificación Bayesiana entre 2 clases ---
%
% class = bayesian_classificate(data, x0, p0, x1, p1, P0, P1)
%
% data: muestras a clasificar (puede ser un escalar o un vector)
% x0/1: espacio creado donde se evalúa 'p'. (p = p(x))
% p0/1: función de densidad de probabilidad de la clase 0/1
% P0/1: Probabilidad a priori de clase 0/1
% class: dato clasificado,
%       class = 0 si clasifica clase 0 (p0(data)*P0 > p1(data)*P1)
%       class = 1 si clasifica clase 1 (p0(data)*P0 < p1(data)*P1)
%
% Ejemplo:
% x0 = linspace(0,10,1e3);
% x1 = linspace(-4, 6, 1e3);
% p0_true = ones(length(x0),1)/10;
% p1_true = normpdf(x1,2,1);
% data = normrnd(2,1);
% bayesian_classificate(data, x0, p0, x1, p1, 0.5, 0.5)
```

Para completar el ejercicio, se generaron entonces 100 nuevas muestras de cada clase y se las clasificó con dicho procedimiento. Sabiendo a que clase pertenecían las muestras se calcularon los errores parciales, esto es la razón entre la cantidad de elecciones equivocadas y el total de las nuevas muestras:

$$P(error|w_1) \simeq \frac{elegir\, clase 2}{total\, de\, muestras} \simeq \frac{errores}{100}$$

y lo mismo en el caso de la clase 2. Luego el error total se calculó como la suma de estos dos errores parciales ponderados por las probabilidades *a priori* de clase correspondientes:

$$\boxed{P(error) = P(error|w_1)P(w_1) + P(error|w_2)P(w_2)}$$

Los resultados obtenidos de las simulación fueron los siguientes:

```
Clasificación Bayesiana
Error total - Parzen: 0.1860
Error total - KNN, k = 1: 0.2980
Error total - KNN, k = 10: 0.1820
Error total - KNN, k = 50: 0.1820
Error total - KNN, k = 100: 0.1780
```

5. e) Implemente la regla de clasificación del K vecino más cercano para $K = 1, 11, 51$ y calcule el error al clasificar las mismas muestras que en d).

Una vez más, basándose en la función "*knnsearch()*" de MATLAB, se escribió una rutina para clasificar según la regla del K vecino más cercano. Dicho procedimiento implica encontrar las 'k' muestras de entrenamiento más cercanas a la observación a clasificar (utilizando la distancia euclidiana) y contando cuantas de ellas pertenecen a cada clase. Luego se clasifica por la mayoría de estos vecinos, i.e. si la mayoría pertenece a la clase 1, entonces se clasifica la clase 1, caso contrario se clasifica la clase 2 (archivo: "knnr_classificate.m").

```
% --- Función de Clasificación KNNR entre 2 clases ---
%
% class = knnr_classificate(data, X0, X1, k)
%
% data: Muestras a clasificar (puede ser un escalar o un vector)
% X0: Muestras de entrenamiento de la clase 0
% X1: Muestras de entrenamiento de la clase 1
% class: Dato clasificado,
%       class = 0 si clasifica clase 0 (de los k vecinos, mayoría de X0)
%       class = 1 si clasifica clase 1 (de los k vecinos, mayoría de X1)
%
% Ejemplo:
% X0 = normrnd(2,1,1e3,1);
% X1 = rand(1e3,1)*10;
% knnr_classificate(rand*10,X0,X1,51)
```

Los resultados obtenidos de la simulación fueron los siguientes

```
Clasificación por regla de los K vecinos más cercanos
Error total - KNNR, k = 1: 0.3600
Error total - KNNR, k = 11: 0.2160
Error total - KNNR, k = 51: 0.1880
```

6. f) Escriba sus conclusiones de las simulaciones realizadas.

Al analizar los datos de los errores obtenidos en la clasificación, lo primero que se hizo fue calcular el error ideal que se obtendría en caso de tener las densidades de probabilidad reales y utilizar el clasificador bayesiano. Como se puede ver en las curvas punteadas de las figuras anteriores, las intersecciones de estas se encuentran en los puntos de $x = 0$, $x = 0{,}1\, y\, x = 3{,}9$ aproximadamente. Esto implica un error compuesto del área debajo de la gaussiana entre 0 y 0.1, el área del rectángulo de 0.1 a 3.9 de altura 1/10*P1, y la cola de la gaussiana de 3.9 a infinito. El resultado de esta cuenta da:

```
Error ideal: 0.1728
```

Este resultado teórico permite comparar los valores (aleatorios) arrojados por las simulaciones con una referencia constante.

En primer lugar, se observa que al estimar por KNN con K=1 y luego clasificar por Bayes, se obtiene un error mucho mayor al resto de los casos, posiblemente debido al exceso de resolución de este método, que como se vio en las figuras anteriores posee una dosis de ruido grande en la estimación.

Por otro lado, en lo que respecta a la regla del K vecino más cercano, se puede ver como el error disminuye a medida que el valor de 'k' aumenta, obteniendo el mejor caso para $k = 51$.

Lo mismo puede decirse, al clasificar por Bayes, luego de haber estimado por KNN, ya que a medida que el valor de 'k' aumenta, el error disminuye, exhibiendo un mínimo con $k = 100$.

Estos resultados fueron confirmados al realizar un ciclo que repita el programa completo reiteradas veces, de forma de obtener errores promedio de los métodos estudiados (archivo: "error_avg.m"). No obstante, para que

la realización sea posible, fue necesario disminuir la cantidad de muestras de entrenamiento generadas ('N') de 10^4 a 10^3, así como también la resolución del muestreo del espacio 'x' para estimar las probabilidades con los distintos métodos. A continuación se detallan los resultados obtenidos:

```
--- Errores promedio ---
Iteraciones: 1000
# Muestras de entrenamiento N: 1000
.
Clasificación Bayesiana
Error promedio - Parzen: 0.1750
Error promedio - KNN, k = 1: 0.2566
Error promedio - KNN, k = 10: 0.1780
Error promedio - KNN, k = 50: 0.1742
Error promedio - KNN, k = 100: 0.1763
.
Clasificación por regla de los K vecinos más cercanos
Error promedio - KNNR, k = 1: 0.2830
Error promedio - KNNR, k = 11: 0.1981
Error promedio - KNNR, k = 51: 0.1815
```

Como se puede ver, en lo que respecta a error total en el caso de la decisión bayesiana, en promedio, todos los métodos se asemejan. Sin embargo, al observar los resultados de la clasificación por regla de KNN, es notable como a medida que aumenta el valor de 'k', el error promedio disminuye acercándose al valor del error en el caso anterior.

Resultados similares se obtuvieron para distinta cantidad de iteraciones y muestras de entrenamiento:

```
--- Errores promedio ---
Iteraciones: 100
# Muestras de entrenamiento N: 10000
.
Clasificación Bayesiana
Error promedio - Parzen: 0.1853
Error promedio - KNN, k = 1: 0.2624
Error promedio - KNN, k = 10: 0.1740
Error promedio - KNN, k = 50: 0.1710
Error promedio - KNN, k = 100: 0.1718
.
Clasificación por regla de los K vecinos más cercanos
Error promedio - KNNR, k = 1: 0.2774
Error promedio - KNNR, k = 11: 0.1922
Error promedio - KNNR, k = 51: 0.1798
```