FACULTAD DE INGENIERIA
Universidad de Buenos Aires

86.55 Teoría de detección y estimación

Trabajo práctico final
Estimación y clasificación no paramétrica

1C 2020

Hsieh, Pablo 97363 hsieh.pablo@gmail.com
phsieh@fi.uba.ar

Resumen

En este documento se presenta el desarrollo, los resultados, y el análisis del estudio de la estimación y clasificación por métodos no paramétricos. Se utilizaron ventanas de parzen y Kn vecinos más cercanos para estimar distribuciones y la regla del K vecino más cercano para clasificar muestras.

1 Enunciado

1.1 Consignas

a) Para las dos distribuciones y probabilidades a priori dadas, generar $N_1 = N_2 = 10^4$ muestras de cada una.

b) Estimar las diferentes distribuciones F_1 y F_2 utilizando Parzen con la ventana dada.

c) Estimar las diferentes distribuciones F_1 y F_2 utilizando Kn vecinos más cercanos para k=1, 10, 50 y 100.

d) Para **b)** y **c)** realizar un clasificador y clasificar 10^2 nuevas muestras, medir el error obtenido.

e) Implementar la regla de clasificación del K vecino más cercano para K=1, 11 y 51 y calcular el error al clasificar las mismas muestras en **d)**.

1.2 Datos para el ejercicio

- $P(w_1) = 0{,}4$
- $P(w_2) = 0{,}6$
- F_1=Gaussiana(1,4) $\Rightarrow \quad \sigma_1 = 2$
- F_2=Gaussiana(4,4) $\Rightarrow \quad \sigma_2 = 2$
- Ventana Gaussiana(μ_w,σ_w^2)

2 Desarrollo teórico

Como el aprendizaje es supervisado, se puede estimar las densidades de las muestras de cada clase: $\hat{p}_n(x) = \hat{p}_n(x, w_i)$.

2.1 Estimación por el método de las ventanas de Parzen

El método de las ventanas de Parzen se basa en estimar la densidad punto a punto dependiendo de las muestras de entrenamiento presentes alrededor. Para ello utiliza se utiliza una ventana, llamada de Parzen, cuya función es ponderar de alguna forma las muestras de entrenamiento presentes alrededor. Esta ventana es típicamente la $\varphi(x)$ rectangular que se presenta a continuación.

$$\varphi(x) = \begin{cases} 1 & |x| \leq \frac{1}{2} \\ 0 & \text{otro} \end{cases} \tag{2.1}$$

Luego, para realizar la ponderación mencionada, se realiza una sumatoria sobre todas las muestras de entrenamiento de la siguiente forma

$$\hat{p}_n(x, w_i) = \sum_{i=1}^{n} \frac{1}{V_n} \varphi\left(\frac{x - x_i}{h}\right) \tag{2.2}$$

donde $V_n = h^d$ siendo d l dimensión del problema. Y $\hat{p}_n(x, w_i)$ es la densidad estimada en el punto x para las muestras de entrenamiento de la clase w_i. Al realizarlo punto a punto sobre todo el soporte, se puede conseguir la estimación de la densidad.

Para el caso de este trabajo, se trabaja con muestras unidimensionales, por lo que $d = 1$ y la (2.2) queda como

$$\hat{p}_n(x, w_i) = \sum_{i=1}^{n} \frac{1}{h} \varphi\left(\frac{x - x_i}{h}\right)$$

Para el caso de h se usará $h = a/\sqrt{N}$, donde N es la cantidad de muestras de entrenamiento y a es una constante arbitraria.

A su vez también se utilizará una ventana gaussiana, por lo que la φ es

$$\varphi_g(x) = \begin{cases} f(x) & |x| \leq \frac{1}{2} \\ 0 & \text{otro} \end{cases} \tag{2.3}$$

donde $f(x)$ es la densidad de la distribución gaussiana; los parámetros de la gaussiana que se utilizará como ventana se explicarán en la sección siguiente.

2.2 Justificación de ventana a utilizar

La ventana a utilizar será gaussiana, de media μ_w y varianza σ_w^2.

La media se la definirá como $\mu_w = 0$ ya que la ventana debe estar centrada en las muestras x_i.

Respecto a la varianza, la idea es tener la menor desviación respecto a la muestra. Como la gaussiana concentra su distribución alrededor de su media, teniendo a más de 2σ respecto de la media el 95 % y más del 99 % si se aleja al menos 3σ, se puede considerar utilizar desvíos mucho mayores a $4\sigma_w$ como la mitad del ancho de la ventana. Esto traerá un resultado de ventana muy "picuda". Por otro lado, la utilizavción de un desvío muy chico da como resultado una ventana que puede llegar a asemejarse a una rectangular. Entonces se analizará por simulación los resultados que traen diferentes valores de σ_w siguiento la relación

$$\frac{h}{2} = n\sigma_w \quad \Rightarrow \quad \sigma_w = \frac{h}{2n} \tag{2.4}$$

De antemano se cree que utilizar $\sigma_w = h/8$ es lo más conveniente, es decir un desvío de $4\sigma_w$ respecto de la media. La justificación es que dará mayor peso a las muestras donde se centre y no a las que están más alejadas, asemejandose a una delta.

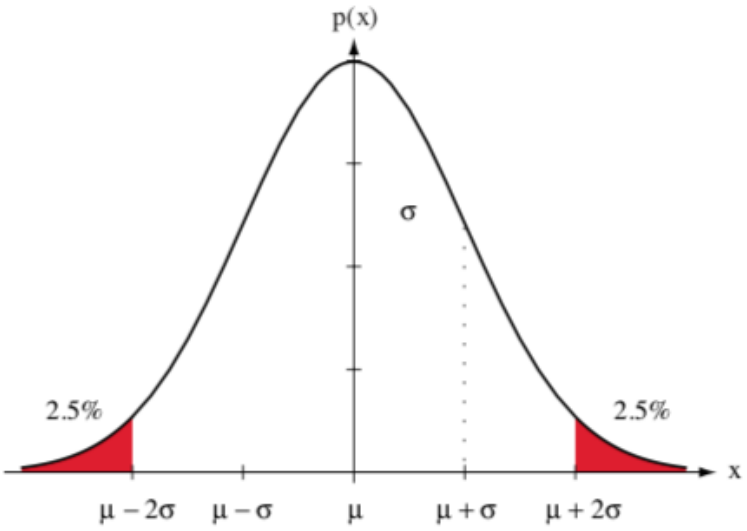


Figura 2.1: Distribución gaussiana.

Se ilustra en la figura 2.2 las diferentes ventanas gaussianas dependientes de la varianza y también la rectangular para contrastar para una longitud de ventana $h = 1$.

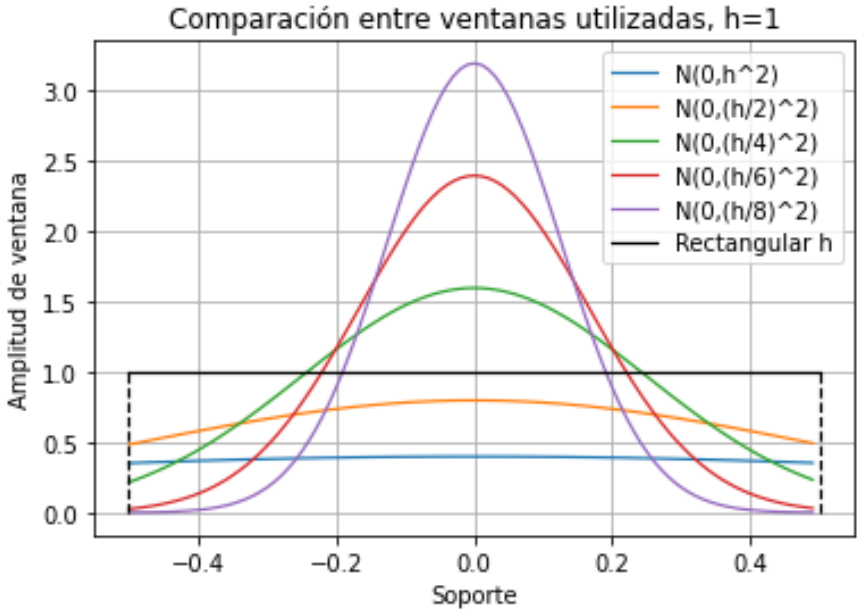


Figura 2.2: Comparación de la ventana gaussiana con la rectangular.

Se puede ver de la figura 2.2 que una ventana gaussiana estándard no difiere en su forma con la rectangular, pero sí en su amplitud, siendo casi el 40 %.

2.3 Estimación por el método de los K_n vecinos más cercanos

La estimación de la densidad por K_n vecinos se basa en armar "cajones" de volumen V_n que encierren las K_n muestras de entrenamiento más cercanas, del total de n muestras de entrenamiento, centrados en un punto x perteneciente al soporte. La expresión del cálculo es la siguiente:

$$\hat{p}_n(x, w_i) = \frac{k_n/n}{V_n} \tag{2.5}$$

entonces, por ejemplo, si se decide utilizar $K_n = 8$ muestras, significa que en cada punto x del soporte se debe encontrar el volumen V_8 que encierre las 8 muestras de entranamiento más cercanas a cada x.

Existe un caso particular que es $K_n = 1$ y centrado en x voy a obtener el volumen V_1 que encierre la muestra de entrenamiento más cercana, por lo que la estimación queda como

$$\hat{p}_n(x, w_i) = \frac{1}{2|x - x_1|} \tag{2.6}$$

donde x_1 es el vecino más cercano. Se puede ver de la ecuación (2.6) que esa estimación no es buena para $\hat{p}(x)$ ya que la integral diverge.

2.4 Clasificación utilizando las densidades estimadas

Se pueden obtener las densidades a posteriori estimadas de las c clases, ya que se conoce la probabilidad de cada clase, entonces se tiene

$$\hat{p}_n(w_i|x) = \frac{\hat{p}_n(x, w_i) \cdot P(w_i)}{\sum_{j=1}^{c} \hat{p}_n(x, w_j)} \qquad i = 1, 2, ..., c \tag{2.7}$$

como este es un problema dicotómico, se tiene

$$\hat{p}_n(w_i|x) = \frac{\hat{p}_n(x, w_i) \cdot P(w_i)}{\hat{p}_n(x, w_1) + \hat{p}_n(x, w_2)} \tag{2.8}$$

Y se clasifica como clase w_1 si $\hat{p}_n(w_1|x) \geq \hat{p}_n(w_2|x)$, es decir si

$$\begin{gathered} \hat{p}_n(w_1|x) \geq \hat{p}_n(w_2|x) \\ \frac{\hat{p}_n(x, w_1) \cdot P(w_1)}{\hat{p}_n(x, w_1) + \hat{p}_n(x, w_2)} \geq \frac{\hat{p}_n(x, w_2) \cdot P(w_2)}{\hat{p}_n(x, w_1) + \hat{p}_n(x, w_2)} \\ \hat{p}_n(x, w_1) \cdot P(w_1) \geq \hat{p}_n(x, w_2) \cdot P(w_2) \end{gathered} \tag{2.9}$$

entonces la regla está dada siguiendo (2.9) y ocurre que

$$\text{si} \quad \hat{p}_n(x, w_1) \cdot P(w_1) \geq \hat{p}_n(x, w_2) \cdot P(w_2) \quad \Rightarrow \quad \text{es de clase } w_1. \tag{2.10}$$

$$\text{si} \quad \hat{p}_n(x, w_1) \cdot P(w_1) < \hat{p}_n(x, w_2) \cdot P(w_2) \quad \Rightarrow \quad \text{es de clase } w_2. \tag{2.11}$$

2.5 Clasificación por la regla de los k vecinos más cercanos

La idea de este método es clasificar una muestra x observando las k muestras de entrenamiento más cercanas a x y luego decidir que x es de la misma clase que la mayoría de las muestras de entranamiento más cercanas. Para ello se necesita que k sea impar ya que en caso de ser par, puede ocurrir que no exista mayoría sino que hay igual cantidad de muestras de entrenamiento de las clases involucradas.

En la figura 2.3 se muestra un ejemplo; alrededor de la muestra x (en este caso el círculo verde) se busca primero los $k = 3$ vecinos más cercanos y luego los $k = 5$ vecinos. Para cada caso el resultado es diferente, cuando $k = 3$ la muestra es de clase triángulo, mientras que cuando $k = 5$ la muestra se la clasifica de clase cuadrado.

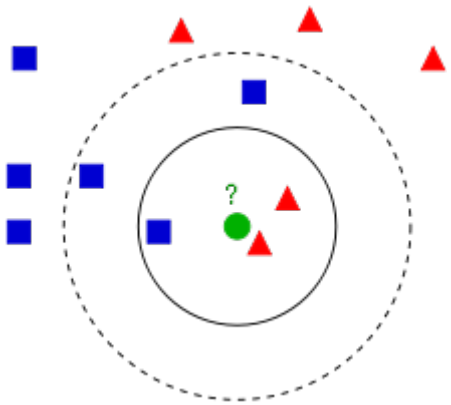


Figura 2.3: Clasificación por método de k vecinos más cercanos en 2 dimensiones.

2.6 Error de clasificación

Teórica

La distribución normal en 1 dimensión está dada por

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \tag{2.12}$$

Para el caso del ejercicio se tienen dos gaussianas de misma varianza y diferente media conocidas. Como se conocen también las probabilidades de clase a priori, se puede obtener la **región de desición** igualando las respectivas funciones discriminantes definidas como

$$g_i(x) = \ln(p(x|w_i)) + \ln(P(w_i)) \tag{2.13}$$

$$g_i(x) = -\frac{1}{2}\ln(2\pi\sigma_i) - \frac{1}{2\sigma_i^2}(x-\mu_i)^2 + \ln(P(w_i)) \tag{2.14}$$

entonces se tienen los discriminantes

$$g_1(x) = -\frac{1}{2}\ln(2\pi 2) - \frac{1}{2\cdot 2^2}(x-1)^2 + \ln(0{,}4)$$
$$g_2(x) = -\frac{1}{2}\ln(2\pi 2) - \frac{1}{2\cdot 2^2}(x-4)^2 + \ln(0{,}6)$$

y al igualar las anteriores se tiene

$$g_1(x) = g_2(x)$$
$$\cancel{-\frac{1}{2}\ln(2\pi 2)} - \frac{1}{2\cdot 2^2}(x-1)^2 + \ln(0{,}4) = \cancel{-\frac{1}{2}\ln(2\pi 2)} - \frac{1}{2\cdot 2^2}(x-4)^2 + \ln(0{,}6)$$
$$-\frac{1}{8}(x-1)^2 + \frac{1}{8}(x-4)^2 = \ln(0{,}6) - \ln(0{,}4)$$
$$-\frac{1}{8}(x^2 - 2x + 1) + \frac{1}{8}(x^2 - 8x + 16) = \ln(0{,}6) - \ln(0{,}4)$$
$$\cancel{-\frac{1}{8}x^2} + \frac{2}{8}x - \frac{1}{8} + \cancel{\frac{1}{8}x^2} + \frac{8}{8}x - \frac{16}{8} = \ln(0{,}6) - \ln(0{,}4)$$
$$\frac{1}{4}x - \frac{1}{8} + x - 2 = \ln(0{,}6) - \ln(0{,}4)$$
$$\frac{5}{4}x = \ln(0{,}6) - \ln(0{,}4) + \frac{1}{8} + 2 = 2{,}530465$$
$$x = 2{,}024372 \tag{2.15}$$

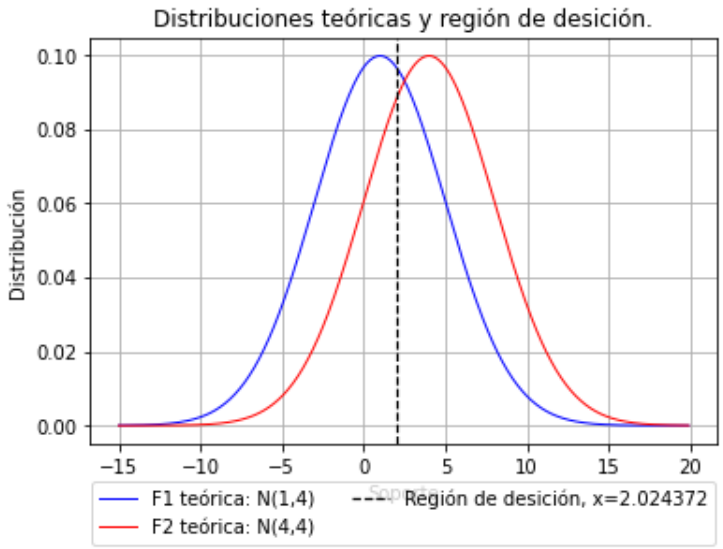


Figura 2.4: Distribuciones teóricas y región de desición teórica.

Es decir que la región de desición está delimitada por $x = x_B$ y se comete error cuando

- una muestra cuya clase real es w_1, cae por encima de x_B. Esto es un error de clasificarla como w_2.
- una muestra cuya clase real es w_2, cae por debajo de x_B. Esto es un error de clasificarla como w_1.

Esto se escribe analíticamente como

$$\begin{aligned} P(\text{error}) &= P(\text{error de clasificación } w_1) + P(\text{error de clasificación } w_2) \\ &= P(\text{clasificar como } w_1, \text{clase real } w_2) + P(\text{clasificar como } w_2, \text{clase real } w_1) \\ &= P(\text{clasificar como } w_1|w_2)P(w_2) + P(\text{clasificar como } w_2|w_1)P(w_1) \end{aligned}$$

$$\Rightarrow \quad P(\text{error}) = p(x < x_B|w_2)P(w_2) + p(x > x_B|w_1)P(w_1) \tag{2.16}$$

Como las distribuciones son gaussianas, se tiene que

$$p(x < x_B|w_2) = \int_{-\infty}^{x_B} \frac{1}{\sqrt{2\pi}2} \cdot \exp\left(\frac{-(x-4)^2}{2 \cdot 4}\right) = 0{,}30426 \tag{2.17}$$

$$p(x > x_B|w_1) = \int_{x_B}^{\infty} \frac{1}{\sqrt{2\pi}2} \cdot \exp\left(\frac{-(x-1)^2}{2 \cdot 4}\right) = 0{,}161622 \tag{2.18}$$

Entonces se obtiene el error teórico

$$P(\text{error}) = 0{,}161622 \cdot 0{,}6 + 0{,}30426 \cdot 0{,}4 = 0{,}218677 \tag{2.19}$$

Implementación algorítmica

Para obtener el error de clasificación se analizó la cantidad de muestras mal clasificadas y se calculó dicha proporción. Para esto es necesario conocer la clase real de la muestra y la clasificación obtenida por cualquiera sea el método.

Se diseñó una estructura de datos como indican las tablas 1 y 2, donde

- la primera columna son las muestras
  - x_1 hasta x_{n1} de clase w_1 generadas con F_1, y n_1 la cantidad de muestras generadas de esa clase.
  - y_1 hasta y_{n2} de clase w_2 generadas con F_2, y n_2 la cantidad de muestras generadas de esa clase.
- la segunda columna es la clase real de la cual provienen las muestras; para el algoritmo se designó como 0 si es de clase w_1 y 1 si es de clase w_2.
- la tercera columna es la clasificación C obtenida, siendo C = {0,1}. Si es clasificado como clase w_1 se designa C = 0 y si es clasificado como clase w_2 se designa C = 1.

Muestra	Clase	Clasificación
x_1	0	C
x_2	0	C
...	...	...
x_{n1}	0	C

Tabla 1: Estructura de datos diseñada, datos de la clase 1.

Muestra	Clase	Clasificación
y_1	1	C
y_2	1	C
...	...	...
y_{n2}	1	C

Tabla 2: Estructura de datos diseñada, datos de la clase 2.

Finalmente se cuenta la cantidad de muestras mal clasificadas con una función que compara la columna 2 con la 3 y cada vez que el valor no es igual se incrementa un contador, aquí llamado "miss_i"; al final se devuelve el contador que es la cantidad de muestras mal clasificadas y se divide por la cantidad real de muestras de esa clase, es decir n_1 o n_2. Esto da respectivamente, el error de clasificación de clase w_1 y w_2.

$$\text{error}_i = \frac{\text{miss\_i}}{n_i} \qquad i = 1, 2 \tag{2.20}$$

Por otro lado también se puede hallar la **región de desición** dada por las densidades estimadas. Para ello también se utiliza la (2.13), pero en vez de las densidades reales se utiliza la estimada, entonces la función discriminante estimada queda como

$$\hat{g}_i(x) = \ln(\hat{p}(x|w_i)) + \ln(P(w_i)) \tag{2.21}$$

es decir que se tienen

$$\hat{g}_1(x) = \ln(\hat{p}(x|w_1)) + \ln(P(w_1))$$
$$\hat{g}_2(x) = \ln(\hat{p}(x|w_2)) + \ln(P(w_2))$$

Luego, se debe hallar la $x = \hat{x}_B$ que satisfaga $\hat{g}_1(x) = \hat{g}_2(x)$, es decir

$$\hat{g}_1(x) = \hat{g}_2(x) \tag{2.22}$$
$$\ln(\hat{p}(x|w_1)) + \ln(P(w_1)) = \ln(\hat{p}(x|w_1)) + \ln(P(w_1)) \tag{2.23}$$

Esto último se realizará de forma numérica y obteniendose una región $x = \hat{x}_B$ estimada. Lo que se hizo fue analizar la diferencia (en módulo) entre $\hat{g}_1(x)$ y $\hat{g}_2(x)$, es decir $\hat{f}(x) = |\hat{g}_1(x) - \hat{g}_2(x)|$ y usar el argumento mínimo de $\hat{f}(x)$ como la región de desición. Esto se consideró ya que es imposible encontrar que $\hat{g}_1(x) = \hat{g}_2(x)$ ya que son estimaciones realizadas de forma numérica, por lo que no va a existir esa igualdad estricta.

Esta forma de resolución será analizada en las secciones correspondientes, sin embargo se cree que puede no estar correcto debido al error numérico.

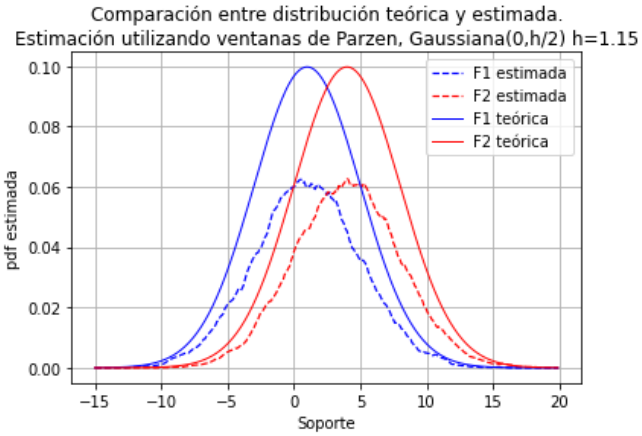
3 Análisis de ventana gaussiana de longitud h y σ_w

Como del enunciado se tienen dos distribuciones gaussianas, $F_1 = \mathcal{N}(1, 4)$ y $F_2 = \mathcal{N}(4, 4)$, se puede asegurar que el soporte de ambas distribuciones juntas estará acotado entre -15 y 20. Esta justificación puede verse de la figura 2.1 y el análisis hecho en la sección 2.2 donde se vio que a más de 3σ la gaussiana cae casi a cero, por lo que a una desviación mucho mayor se puede asegurar que la probabilidad de muestras fuera de esa región es casi nula. Definiendose una desviación de 8σ alrededor de cada media se llega a un soporte contenido en el rango dicho de -15 y 20. Se simuló en ese soporte y con un paso de 0,1 para $N = 10^4$ muestras de entrenamiento.

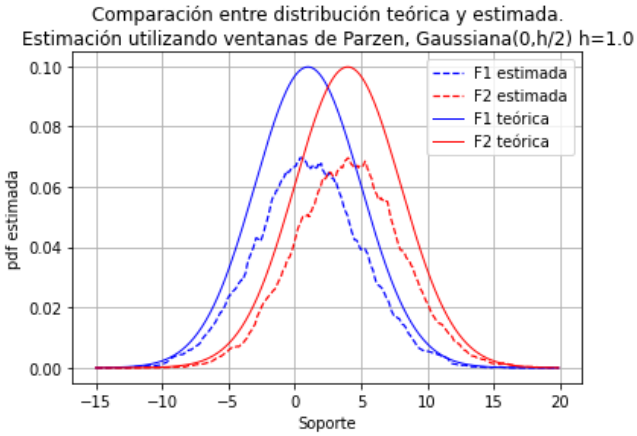
Como se explicó en la sección 2.2 se utilizará una ventana gaussiana de media 0 y desvío σ_w^2. Esta ventana gaussiana estará contenida en una longitud h.

Se simularon los siguientes valores de $h = a/\sqrt{N}$, con $a = \{115, 100, 80, 50, 25\}$ variable y $N = 10^4$, entonces $h = \{1{,}15, 1, 0{,}8, 0{,}5, 0{,}25\}$.

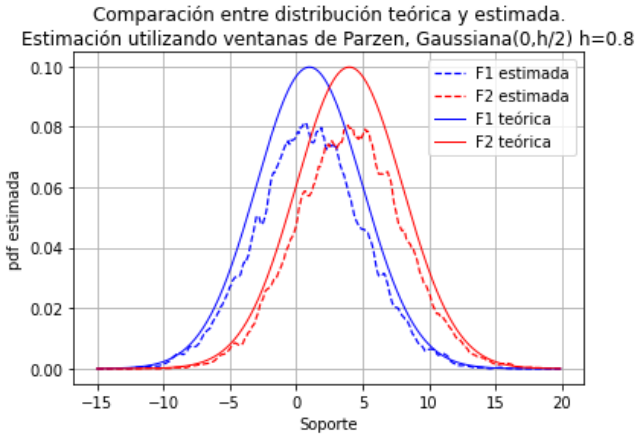
En las figuras 3.1, 3.2, 3.3 y 3.4 se muestran estimaciones realizadas para distintas longitudes de ventana h y desvío $\sigma_w = h/2$, $\sigma_w = h/4$, $\sigma_w = h/6$ y $\sigma_w = h/8$ respectivamente.



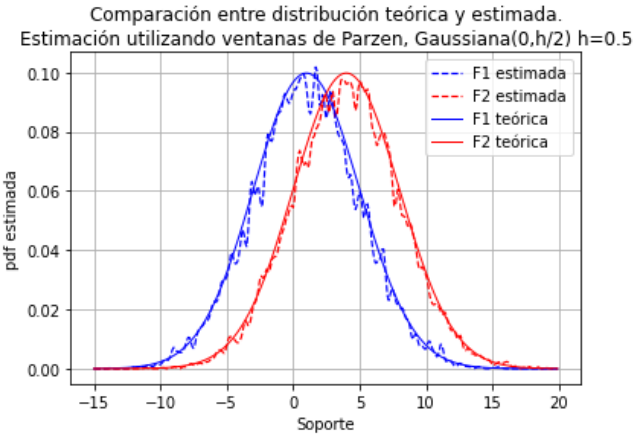
(a) $h = 1{,}15$.



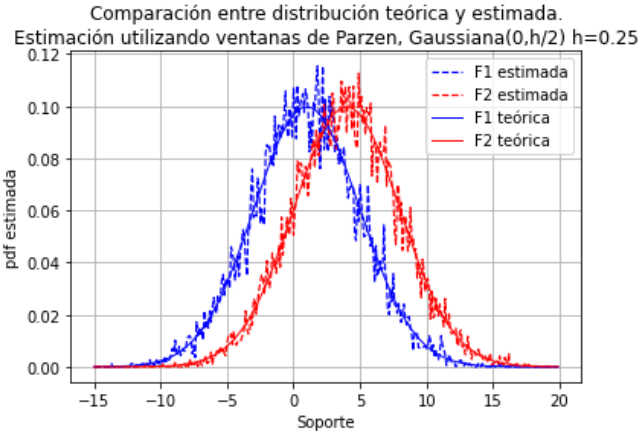
(b) $h = 1$.



(c) $h = 0{,}8$.



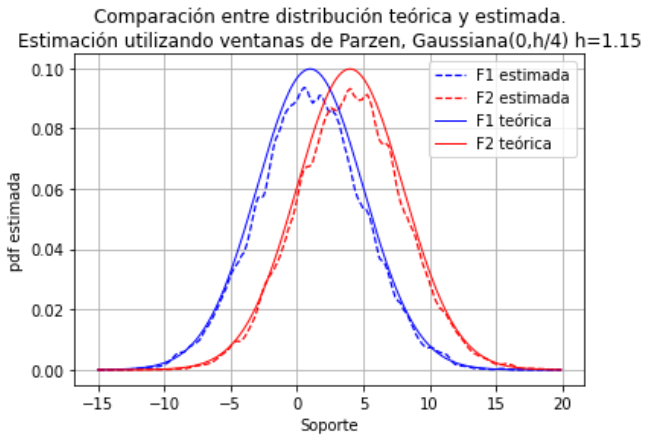
(d) $h = 0{,}5$.



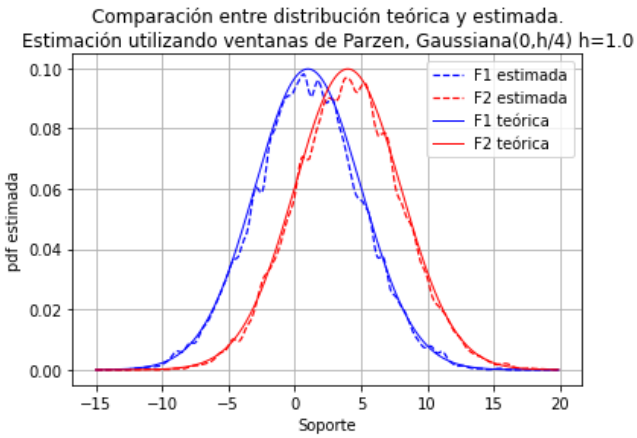
(e) $h = 0{,}25$.

Figura 3.1: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/2$ y diferentes h.

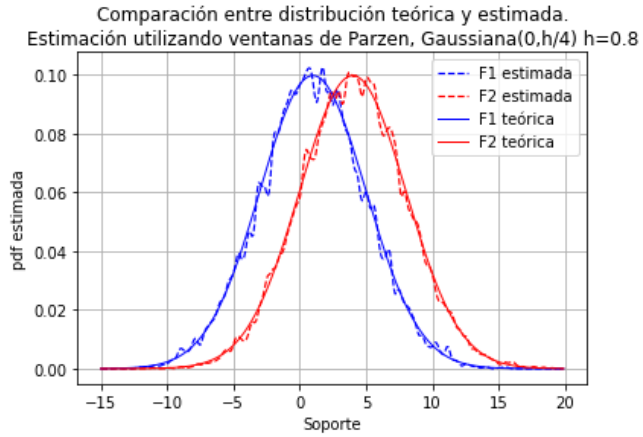
Se puede ver de los resultados en la figura 3.1 que a medida que h disminuye, la estimación se acerca más a la altura de la distribución real, esto verifica que la ventana usada responder correctamente a una distribución de probabilidad ya que al achicar h debe aumentar la altura de la ventana. Por otro lado se ve que la mejor estimación se da con $h = 0{,}5$, y a medida que disminuye empieza a ser mucho más ruidosa.



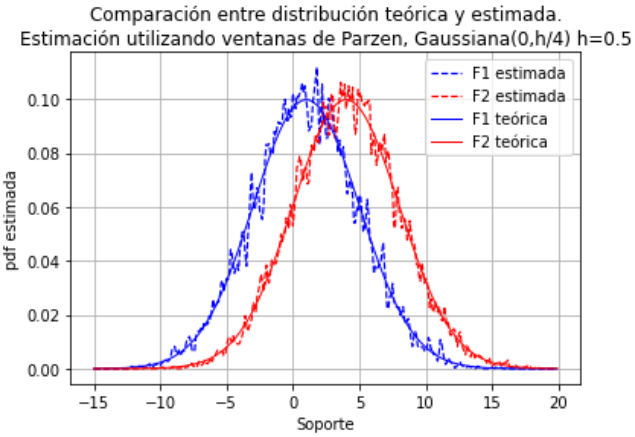
(a) $h = 1{,}15$.



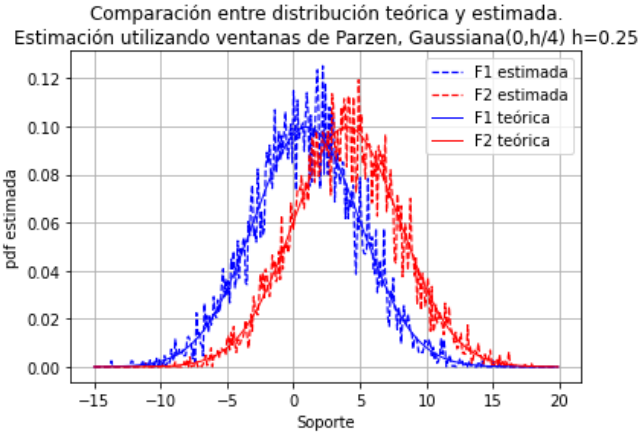
(b) $h = 1$.



(c) $h = 0{,}8$.



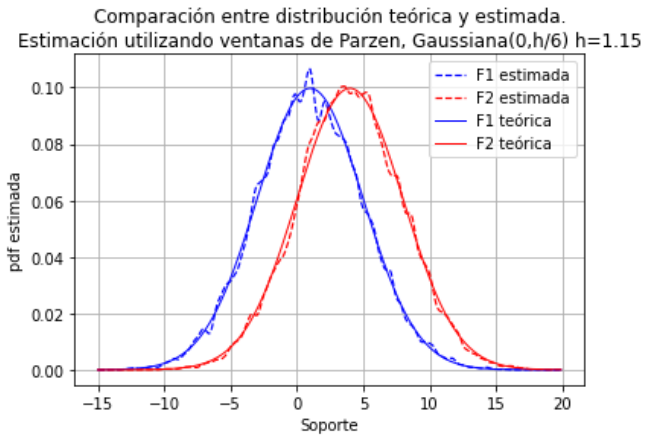
(d) $h = 0{,}5$.



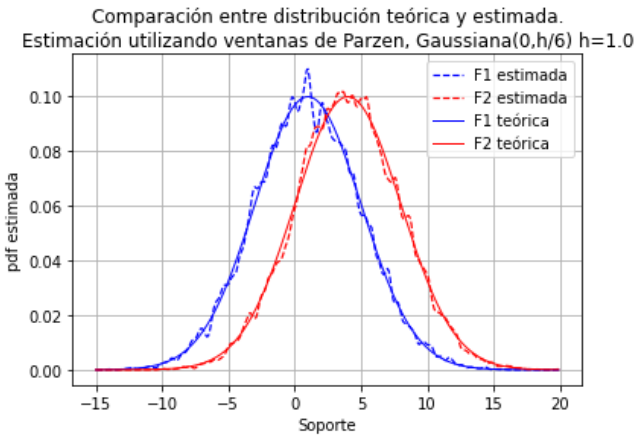
(e) $h = 0{,}25$.

Figura 3.2: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/4$ y diferentes h.

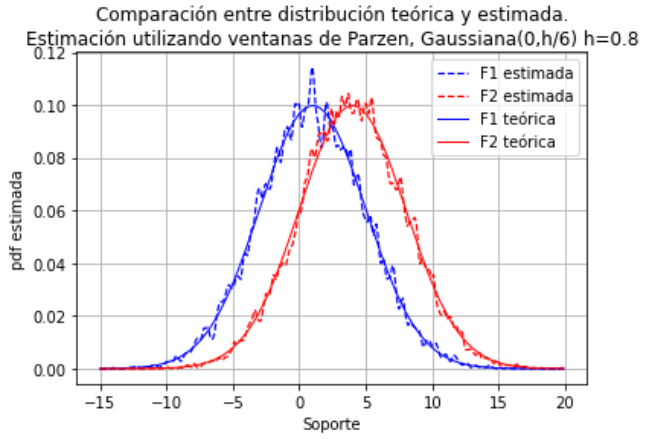
De la figura 3.2 se ve que al usar un σ_w menor al anterior, la estimación se acerca más en amplitud a la distribución real, esto es lógico ya que en la ventana, la gaussiana es más angosta. Se ve que la mejor estimación se da con $h = 0{,}8$ y disminuyendo h por debajo de 0,5 empieza a tener mayor ruido. Es posible que si se use un h entre 0,8 y 1 se pueda conseguir un buen resultado de estimación.



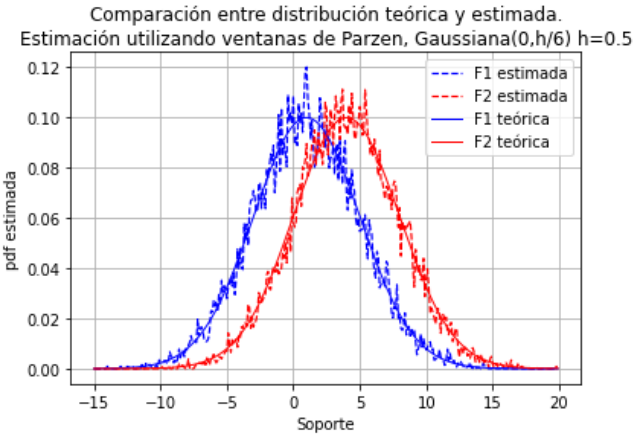
(a) $h = 1{,}15$.



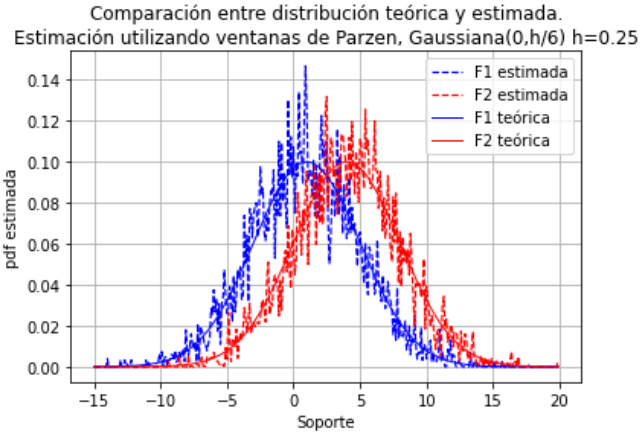
(b) $h = 1$.



(c) $h = 0{,}8$.



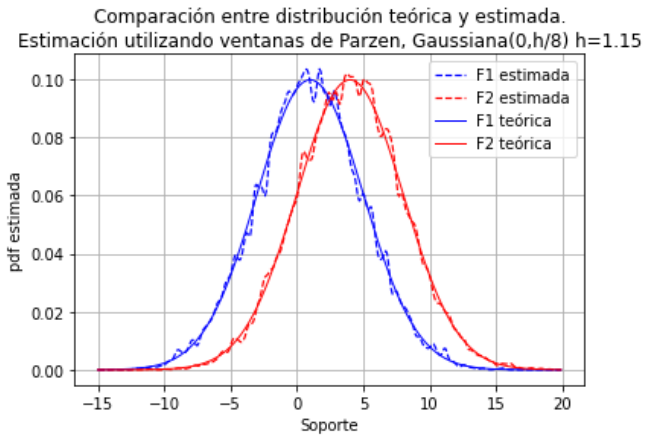
(d) $h = 0{,}5$.



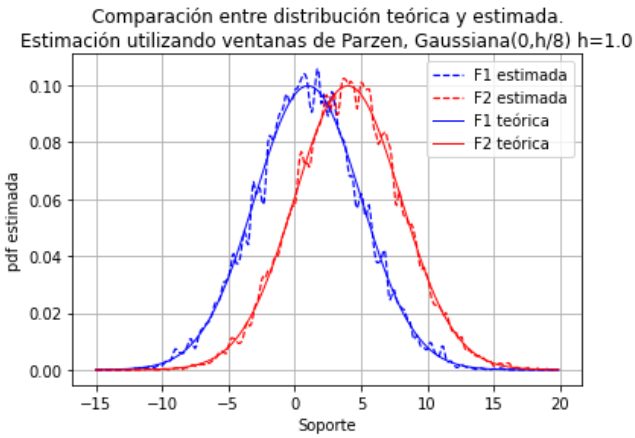
(e) $h = 0{,}25$.

Figura 3.3: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/6$ y diferentes h.

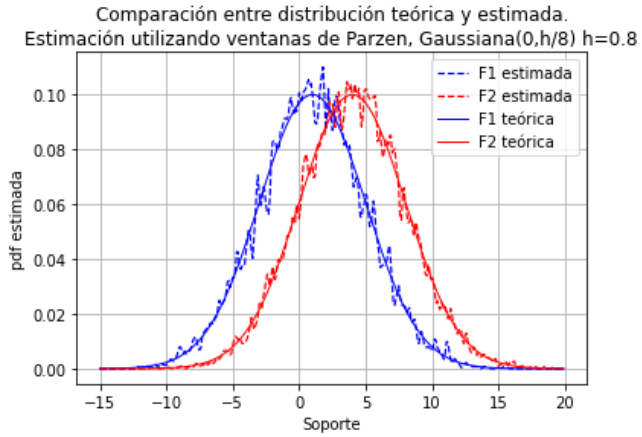
Viendo la figura 3.3.a se ve que la estimación es bastante suave a la distribución real, y a medida que se disminuye la longitud de ventana h a partir de $h = 1$, empieza a aumentar el ruido de forma más fuerte.



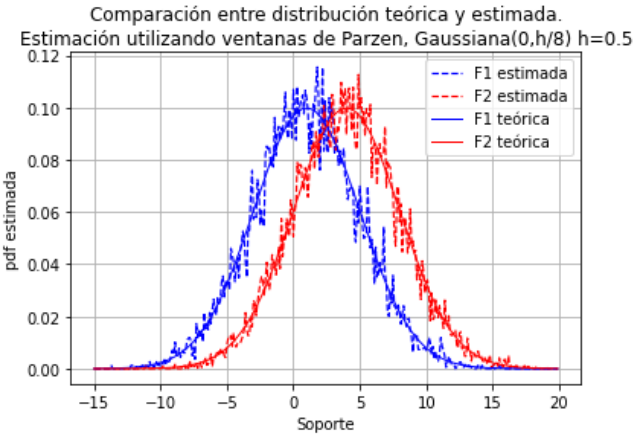
(a) $h = 1{,}15$.



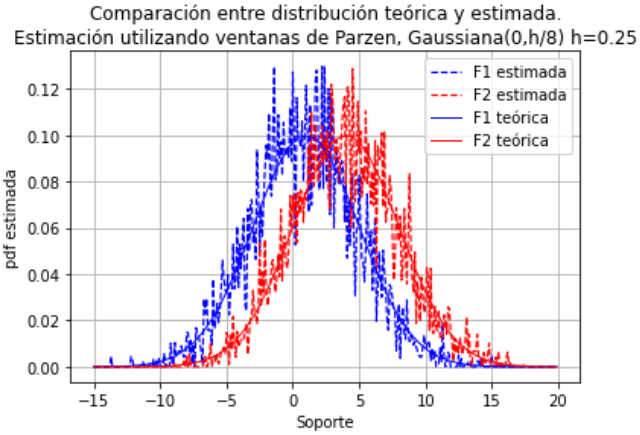
(b) $h = 1$.



(c) $h = 0{,}8$.



(d) $h = 0{,}5$.



(e) $h = 0{,}25$.

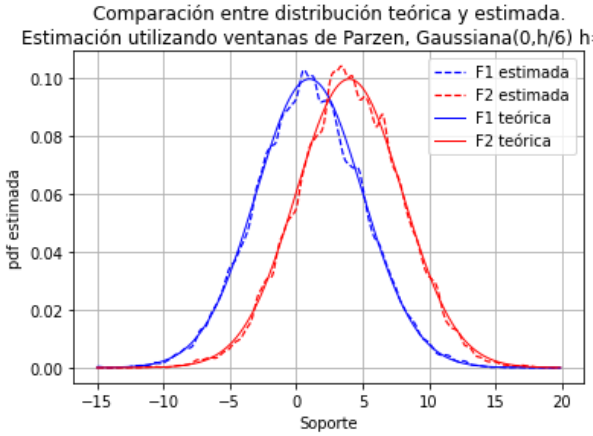
Figura 3.4: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/8$ y diferentes h.

Finalmente de la figura 3.4 se ve que empieza a existir un poco de ruido con el primer $h = 1{,}15$ y a medida que se disminuye h el ruido se hace mucho más intenso.

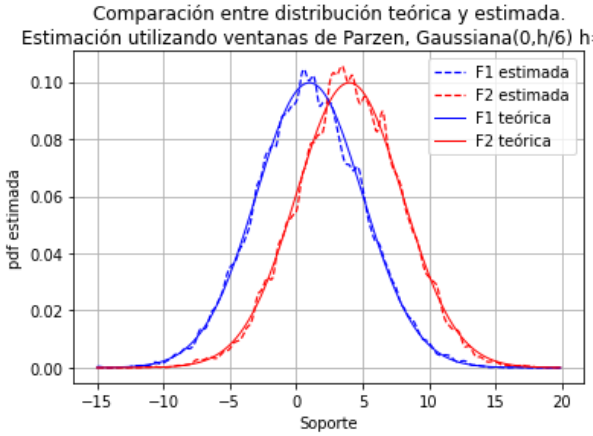
Con lo observado anteriormente de las figuras 3.1, 3.2, 3.3 y 3.4 se ve que es conveniente utilizar alguna de las siguientes configuraciones:

- $\sigma_w = h/4$ y $h = a/\sqrt{10^4}$ con a entre 80 y 100.
- $\sigma_w = h/6$ y $h = a/\sqrt{10^4}$ con a alrededor de 115.

Se simuló con las configuraciones de parámetros previamente mencionados para poder comparar y definir el valor final de σ_w y h. Se muestra en las figuras 3.5 y 3.6 los resultados.

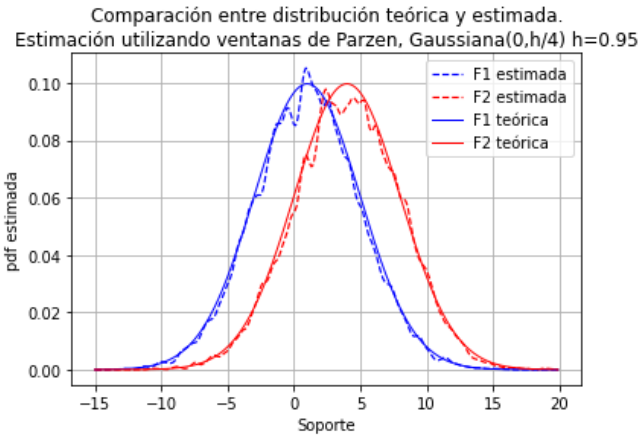


(a) $h = 1{,}2$.

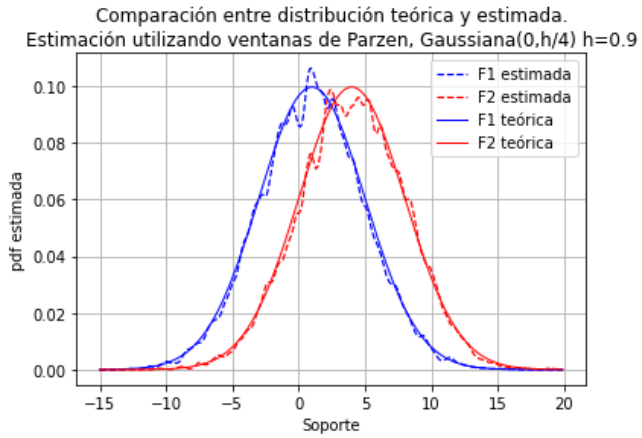


(b) $h = 1{,}1$.

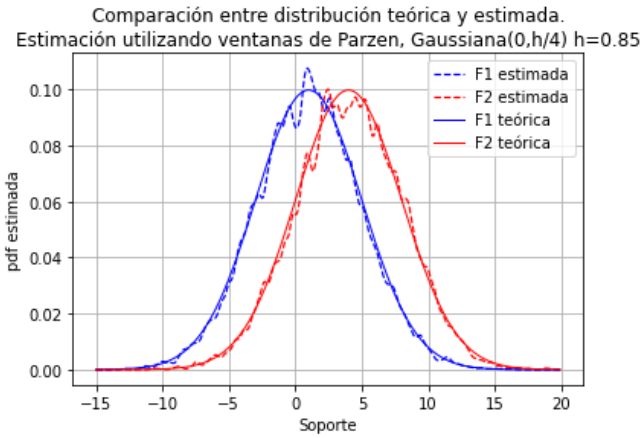
Figura 3.5: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/6$ y $h \in (1; 1{,}2)$.



(a) $h = 0{,}95$.



(b) $h = 0{,}90$.



(c) $h = 0{,}85$.

Figura 3.6: Estimaciones obtenidas por ventanas de Parzen para $\sigma_w = h/4$ y $h \in (0{,}8; 1)$.

Se puede ver que el mejor resultado se dio en la estimación con $\sigma_w = h/6$ frente a los de $\sigma_w = h/4$ ya que presentan estimaciones más suaves. Luego con $\sigma_w = h/6$, cuando $h = 1{,}2$ las variaciones de la estimación son menores que cuando $h = 1{,}1$ por lo tanto se decide utilizar $\sigma_w = h/6$ y $h = 1{,}1$.

4 Resolución

Las simulaciones se realizaron en un soporte entre -15 y 20 y con un paso de 0,1, es decir con un soporte de 350 puntos equiespaciados entre -15 y 20. (La justificación del soporte fue explicada en la sección 3)

4.1 a)

Para las dos distribuciones y probabilidades a priori dadas, se generaron $N_1 = N_2 = 10^4$ muestras de cada una, para las distribuciones $F_1 = \mathcal{N}(1, 4)$ y $F_2 = \mathcal{N}(4, 4)$. Se muestra en la figura 4.1 el resultado obtenido en un histograma clásico standard.

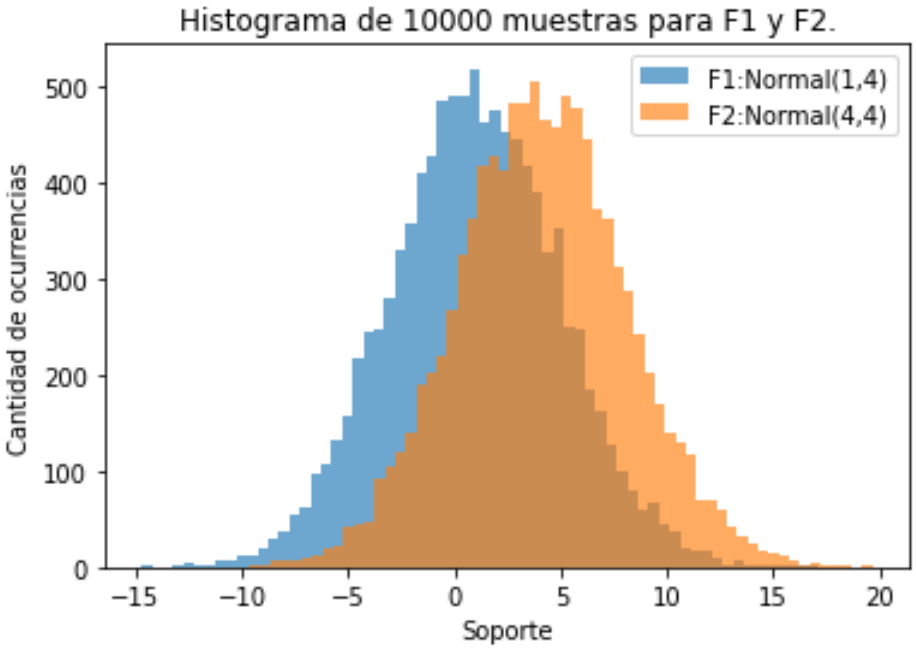


Figura 4.1: Muestras generadas.

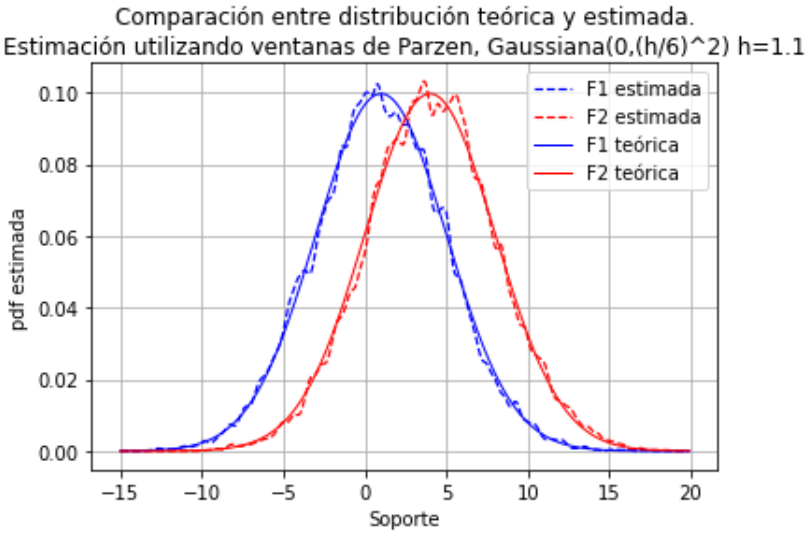
Viendo la figura 4.1 se ve que el histograma tiene claramente la forma de las gaussianas utilizadas. Con mayor cantidad de muestras se puede tener mejor resolución y no tener picos ni valles de más como se puede ver en el resultado.

4.2 b)

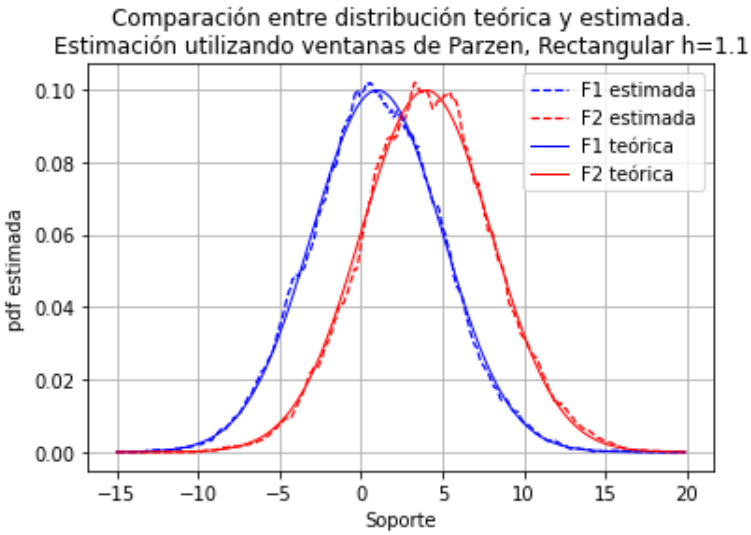
Con los desarrollos de las secciones 2.2 y 3 se estimó la densidad con las muestras generadas en el inciso anterior. Se realizó con ventanas de parzen gaussiana como pedía el enunciado y también se lo hizo con una ventana rectangular para poder comparar y estudiar la incidencia de la ventana utilizada.

La ventana gaussiana usada fue de media 0, desvío $h/6$ y longitud $h = 1{,}1$. La ventana rectangular también fue de longitud $h = 1{,}1$. Los resultados se muestran en la figura 4.2.

Se puede ver de la figura 4.2 que ambos resultados siguen a la densidad real teórica, sin embargo se ve que para la ventana gaussiana se producen variaciones más abruptas que la rectangular. Esto es debido principalmente a la amplitud que tienen las ventanas; en la figura 2.2 de la sección 2.2 se mostró que la ventana rectangular está, en amplitud, por debajo de la gaussiana utilizada, es decir la que tiene desvío de $h/6$. Esto es lo que trae las variaciones abruptas, en comparación con la rectangular, que se ven en esta simulación.



(a) Gaussiana



(b) Rectangular

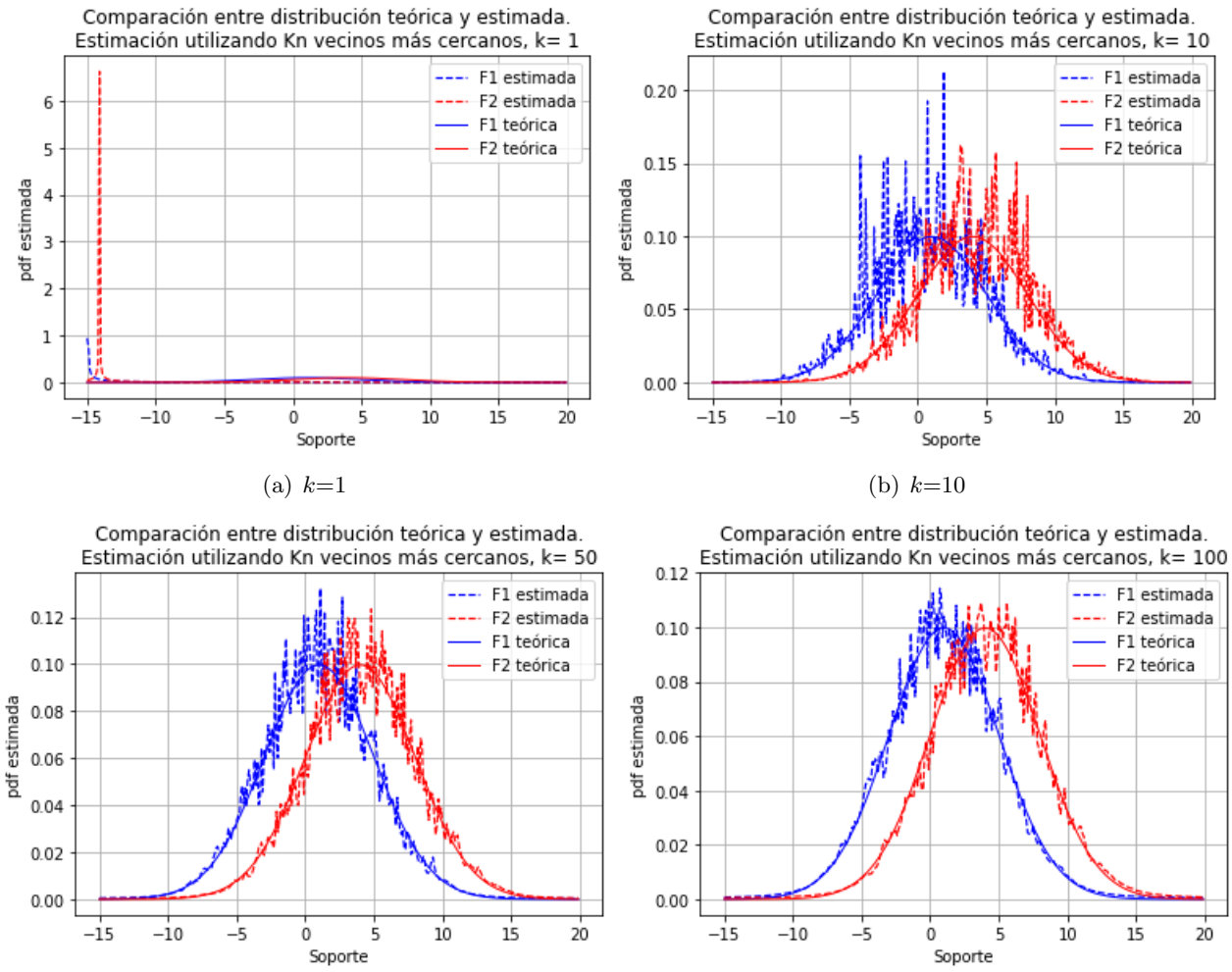
Figura 4.2: Estimación con ventanas de Parzen.

Por otro lado, debido a las formas de las ventanas, la rectangular toma con el mismo peso a la muestra donde está centrada y en los alrededores donde la longitud de ventana alcance; en cambio la ventana gaussiana tiene más peso en la muestra y menos peso en sus alrededores. Por lo tanto si se comparan los resultados de la figura 4.2 con las muestras obtenidas en la figura 4.1 se puede ver que la utilización de la ventana gaussiana estima mejor los picos y variaciones que presentan las muestras, a diferencia de la rectangular que lo realiza de forma más suave.

Lo último dicho tiene ventajas y desventajas. Por un lado, conociendo la distribución teórica, se sabe que al ser gaussiana, la mejor estimación es utilizando la ventana rectangular ya que es más suave y no presenta picos más abruptos. Pero por otro lado, si el aprendizaje fuera no paramétrico, la ventana gaussiana tendría mejor resolución ya que estimaría mucho mejor las variaciones abruptas pudiendo dar una mejor estimación.

4.3 c)

Se estimó la densidad de las muestras generadas con el método de K_n vecinos más cercanos. Se utilizó $n = 1$, 20, 50 y 100, es decir que se estimó con volumenes que ocupen las $K_n = 1$, 20, 50 y 100 muestras de entrenamiento más cercanas a cada punto del soporte. Los resultados se ilustran en la figura 4.3.



(a) k=1

(b) k=10

(c) k=50

(d) k=100

Figura 4.3: Estimación con el método de K_n vecinos más cercanos.

Como se explicó en la sección 2.3, la estimación se basa en la ecuación (2.5) y para el caso particular de $K_1 = 1$ se usa la (2.6).

Se ve que cuando se utiliza 1 solo vecino, la estimación es terriblemente mala, no tiene resolución, tiene un único pico, no se puede apreciar la distribución teórica y, se intuye, no sirve para absolutamente nada.

Sin embargo a medida que se incrementa la cantidad de vecinos, se puede ver que la estimación empieza a aproximarse a la distribución real. Y cuanto mayor cantidad de vecinos se utilicen, menor cantidad de picos y ruido existe.

4.4 d)

Luego de las estimaciones realizadas anteriormente, se simularon 100 muestras nuevas, de las cuales hay de clase w_1 con probabilidad $P(w_1) = 0{,}4$ y de clase w_2 con probabilidad $P(w_2) = 0{,}6$. En este caso sencillo, fueron 40 muestras pertenecientes a la clase $w_!$ y 60 de la clase w_2.

Luego se clasificaron las muestras obtenidas según el desarrollo de la sección 2.4 utilizando las reglas dadas por la (2.10) y (2.11).

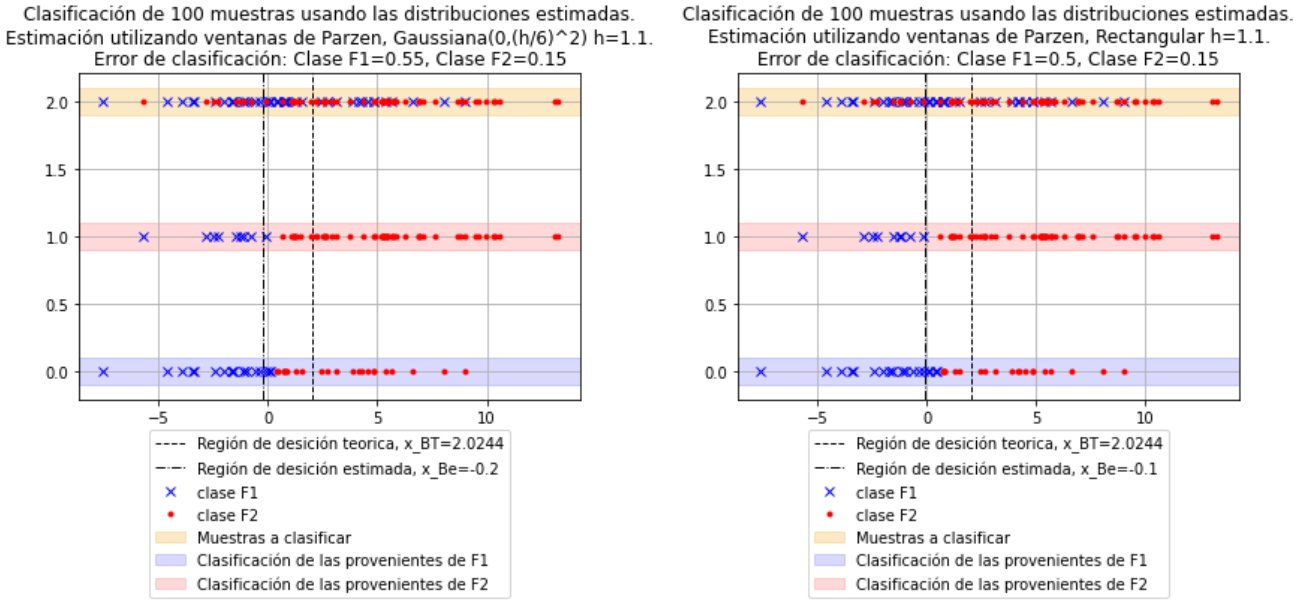
Tanto las muestras generadas, como la clasificación realizada fueron graficadas y se computó el error de clasificación para poder analizar los resultados. La forma de graficar fue la siguiente:

- Las muestras provenientes de clase w_1 son las generadas por la distribución F_1 y fueron marcadas con una cruz azul.

- Las muestras de clase w_2 son las generadas con la distribución F_2 y fueron señalizadas con un punto rojo.
- Las muestras reales se las graficaron sobre el eje x con ordenada $y = 2$ y recuadradas en fondo amarillo.
- La clasificación realizada con las muestras provenientes de la distribución F_1 se las graficó con ordenada $y = 0$ y en fondo azul.
- La clasificación de las muestras provenientes de F_2 se las graficó con ordenada $y = 1$ y en fondo rojo.
- En línea punteada - - se graficó la región de desición teórica obtenida en la ecuación (2.15).
- En línea punteada con puntos intercalados -.-. se graficó la región de desición a partir de las estimaciones realizadas, según el desarrollo de la ecuación (2.23)

El error fue realizado como se explica en la sección 2.6.

A continuación se presenta en la figura 4.4 las clasificaciones obtenidas con las densidades estimadas por el método de ventanas de Parzen.



(a) Gaussiana

(b) Rectangular

Figura 4.4: Clasificación realizada con la estimación por ventanas de Parzen.

Se puede ver de la figura 4.4 que la región de desición teórica x_B no resultó ser la región que realiza la clasificación, sin embargo está cercana a la región de clasificación estimada $\hat{x}_B$. Se puede ver también que para las muestras menores a $\hat{x}_B$ se las clasifica como clase F_1 y las superiores como clase F_2 y es correcta la regla de clasificación debido a la ubicación de las densidades estimadas.

Se atribuye la diferencia entre x_B con $\hat{x}_B$ a la estimación realizada, la presencia de una curva que no es suave sumado a ser una expresión numérica y no analítica da por resultado que la simulación no corresponda con la teoría.

Viendo los errores se puede ver que la clase w_1 presenta error de clasificación mayor que la clase w_2; y se obtiene el error total de clasificación a partir de la estimación de la siguiente forma

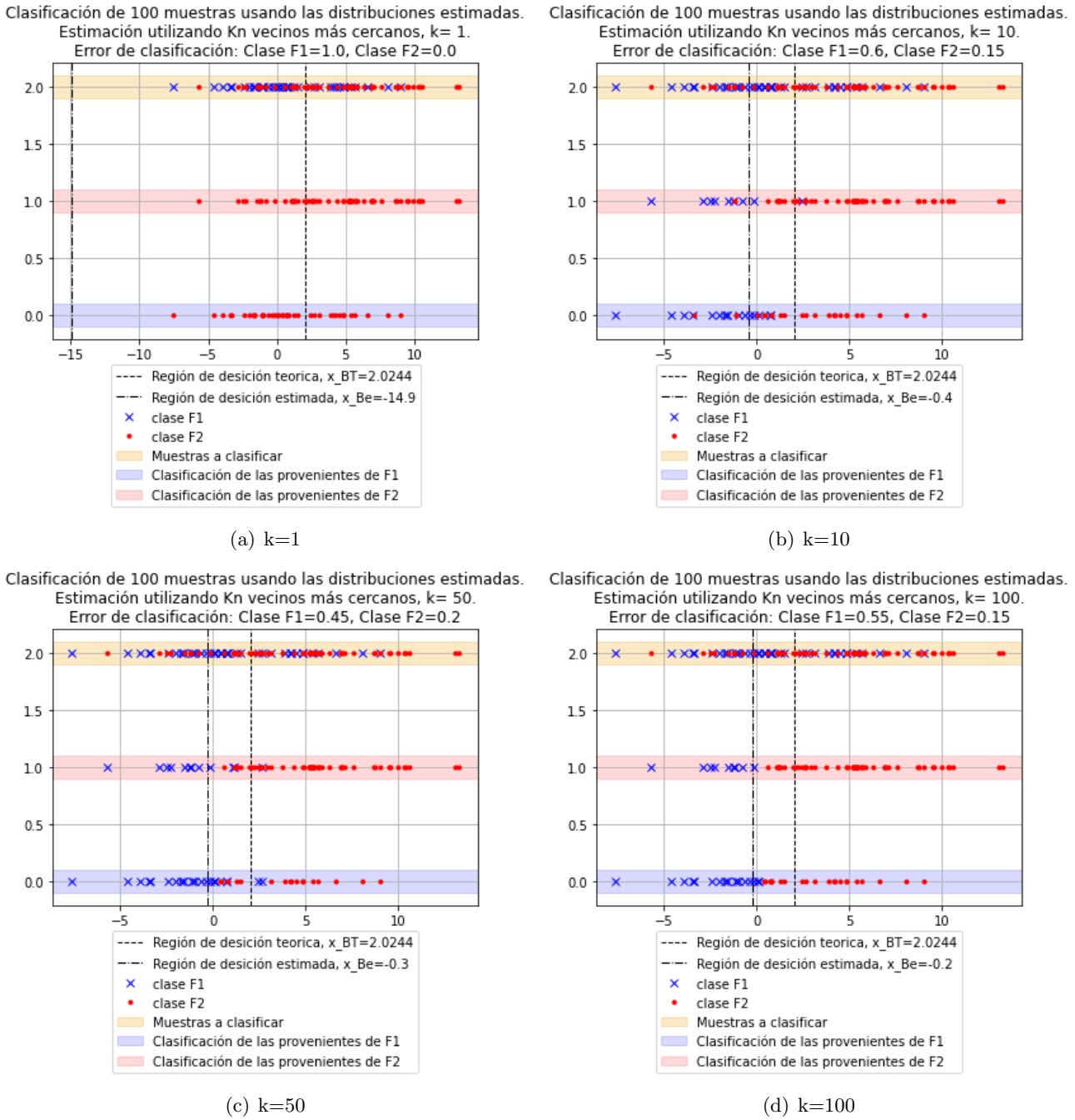
$$\hat{p}(\text{error}) = \hat{p}(\text{error clase } w_1)P(w_1) + \hat{p}(\text{error clase } w_2)P(w_2) \tag{4.1}$$

Se resume en la tabla 3 el error obtenido al clasificar con las densidades estimadas por ventanas de parzen.

Ventana	error dado por clase		$\hat{p}$(error)
	Clase w_1	Clase w_2	
Gaussiana	0,55	0,15	0,31
Rectangular	0,5	0,15	0,29

Tabla 3: Error obtenido al clasificar con ventana de parzen.

En la figura 4.5 se muestran los resultados de la clasificación dada las densidades estimadas por K_n.



(a) k=1

(b) k=10

(c) k=50

(d) k=100

Figura 4.5: Clasificación realizada con la estimación por K_n vecinos más cercanos.

Se puede ver de los resultados presentados en la figura 4.5 que cuando $k = 1$ la clasificación siempre da de clase w_2 para muestras mayores a $\hat{x}_B > -14{,}9$, y como las muestras no son siquiera inferiores a $x = -10$ siempre son clasificadas como de clase w_2. Esto tiene sentido al ver la estimación dada en la figura 4.3.a ya que esa densidad estimada fue de dos picos con centros en aproximadamente $x = -15$ pero siendo la clase w_2 con media mayor que la clase w_1. Y para los otros valores de k, los resultados

resultan más coherentes. También se verifica que la $\hat{x}_B$ calculada delimita correctamente las regiones de cada clase.

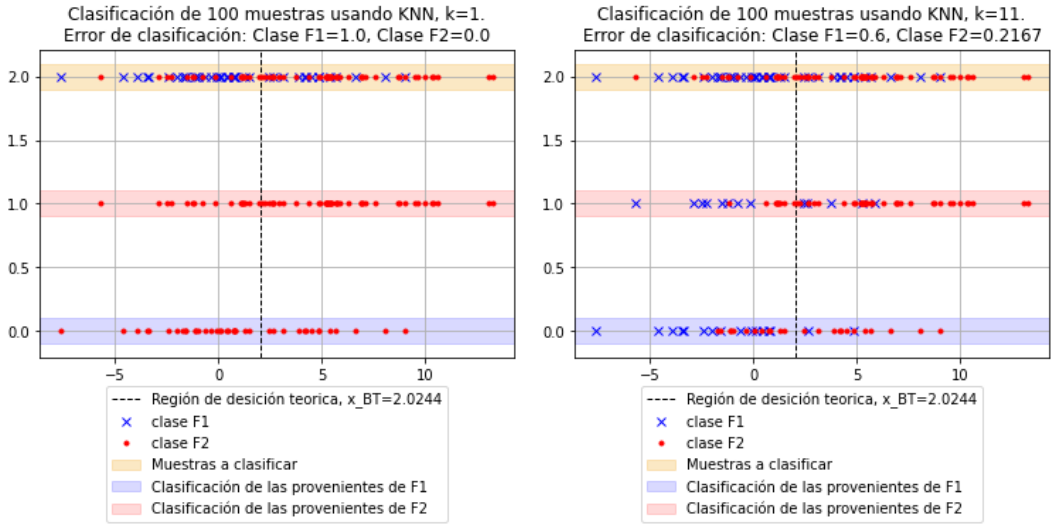
Por último, se observa que hay una tendencia de disminuir el error a medida que k aumenta. Al igual que antes, se resumen los errores obtenidos en la tabla 4 y se calculó el error total de estimación con la expresión (4.1).

K_n	error dado por clase		$\hat{p}$(error)
	Clase w_1	Clase w_2	
1	1	0	0,4
10	0,6	0,15	0,33
50	0,45	0,2	0,3
100	0,55	0,15	0,31

Tabla 4: Error obtenido al clasificar con K_n vecinos más cercanos.

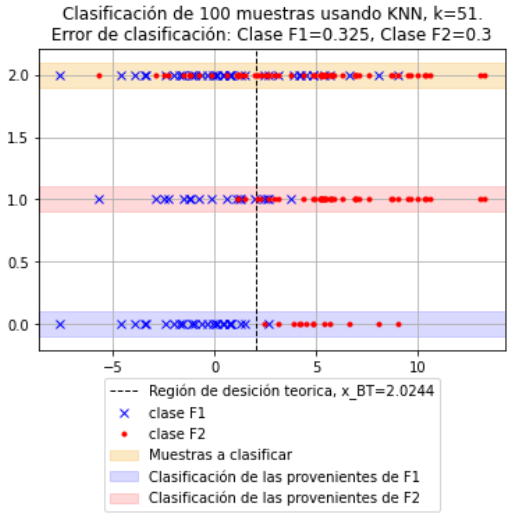
4.5 e)

Finalmente, se implementó la regla de desición de los k vecinos más cercanos. Se presentan los resultados en la figura 4.6.



(a) k=1

(b) k=11



(c) k=51

Figura 4.6: Clasificación de muestras por la regla de los k vecinos más cercanos.

Se puede ver que cuando $k = 1$ vuelve a tener problemas, o mejor dicho que la clasificación realizada es siempre la misma. Esto resulta extraño ya que implica que para todo el soporte, cada muestra que se quiere clasificar tiene el vecino más cercado de clase w_2 lo cual no parece ser algo razonable. Y en caso de que fuera una ocurrencia aislada se puede considerar válida, sin embargo todas las simulaciones que se obtuvieron dan el mismo resultado cuando $k = 1$, por lo que se intuye que es un problema de modelo cuando $k = 1$. Por otro lado, si la implementación hubiera estado mal realizada, cuando se usa otros k también debería tener problemas, sin embargo no es el caso.

Entonces descartando $k = 1$ y analizando $k = 11$ y $k = 51$ se puede ver que la región de desición no implica demasiado peso para $k = 11$ sin embargo para $k = 51$ se ve que la clasificación se "acomoda" alrededor de la región teórica x_B. Esto demuestra que a medida que se utiliza más vecinos, la clasificación se acercará más al caso teórico.

Por otro lado se ve que el error de clasificación también disminuye, siendo de $0{,}6 * 0{,}4 + 0{,}2167 * 0{,}6 = 0{,}37$ para $k = 11$ y de $0{,}325 * 0{,}4 + 0{,}3 * 0{,}6 = 0{,}31$ para $k = 51$, por lo que el error es menor al usar más vecinos para clasificar. Esto es lógico ya que si se utiliza mayor cantidad de vecinos se puede conocer estadísticamente mejor a la muestra que se quiere clasificar.

K	error dado por clase		$\hat{p}$(error)
	Clase w_1	Clase w_2	
1	1	0	0,4
11	0,6	0,2167	0,37
51	0,325	0,3	0,31

Tabla 5: Error obtenido al clasificar con la regla de K vecinos más cercanos.

4.6 Resumen de errores obtenidos

Se resumen los errores obtenidos para poder compararlos en la tabla 6.

Clasificación		error
Teórica		0.22
Parzen	gaussiana	0,31
	rectangular	0,29
K_n	k=1	0,4
	k = 10	0,33
	k = 50	0,3
	k=100	0,31
KNN	k=1	0,4
	k = 11	0,37
	k=51	0,31

Tabla 6: Resumen de los resultados de errores dados por clasificación obtenidos por diferentes métodos.

Se puede ver que el error bayesiano teórico es el menor de todos, verificándose lo hablado en las clases acerca de ser el error óptimo y el menor que se puede obtener.

Se vuelve a remarcar el problema cuando $k = 1$ teniendo el peor error y siendo lo peor que se puede utilizar.

Existe una tendencia general cuando se aumenta la cantidad de vecinos más cercanos, en la regla de desición KNN, o cuando se aumenta la cantidad de K_n vecinos utilizados para estimar la densidad ya que los resultados de clasificación son mejores al verse disminuído el error.

Viendo los mejores casos de cada método se puede ver que los errores obtenidos al clasificar difieren en menos del 1 % entre ellos (Parzen con ventana rectangular 0,29; K_n=50 vecinos 0,3; KNN con 51 vecinos 0,31) concluyendose que la implementación realizada es satisfactoria.

5 Conclusiones

Se pudo estudiar de forma práctica los métodos de estimación no paramétricos y familiarizarse con ellos utilizandose las bases teóricas estudiadas en clase. Es muy útil tener una herramienta extra que no sea el histograma estándard, además se vio que los resultados obtenidos fueron mejores que el histograma clásico, por lo que si existen muestras provenientes de distribuciones más complicadas, probablemente no alcance con solo estimar utilizando un histograma estándard y se requerirá convocar a los métodos estudiados.

Se logró estudiar y analizar la dependencia al utilizar distintas ventanas para estimar por el método de parzen; en particular la rectangular unitaria y la gaussiana de media nula y varianza variable dependiente de la longitud de ventana. Se vio su influencia sobre la estimación dando mayor o menor resolución. También se logró estudiar la dependencia de la ventana utilizada en función de su ancho h, concluyendose que si el ancho es mayor se puede tener mejor resolución, y cuando el ancho es menor existe el llamado "efecto peine" que provoca mucho ruido. Siempre se debe analizar la dependencia de la longitud ya que puede traer los efectos extremos mencionados.

A la hora de estimar con K_n vecinos, se vio el problema de utilizar k chicos, en particular cuando se utiliza solo 1 vecino para estimar la densidad. Cuando se la aumenta, el ruido empieza a disminuir.

Se logró ver que la regla de clasificación de los k vecinos más cercanos es muy útil y no se necesita conocer las densidades. Esto puede traer una gran ventaja si la aplicación que se desea implementar es solo de clasificar y no conocer la forma paramétrica del origen de las muestras.

Se verificó que el mejor error que se puede obtener es el bayesiano, siendo la regla óptima de desición.

6 Bibliografía

- Duda, Hart, Stork, *Pattern classification*, 2nd ed (2001).
- Apuntes de clases
- Documentación de python.

7 Código implementado

Se implementó el siguiente código en python para la resolución del trabajo práctico. Está comentada la explicación de las funciones y de los procedimientos armados.

```
# Teoria de deteccion y estimacion - FIUBA
# 1er cuatrimestre de 2020

# Autor: PABLO HSIEH
# Padron: 97363

# Estimacion no parametrica - PARZEN, KN, Regla de desicion KNN


import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Gaussiana ------------------------------------------------------------------
def gaussiana(x,mu,sigma):
#Es la pdf gaussiana
#Para el caso del error se usa mu=0, sigma=1
    return (np.exp( (-(x-mu)**2)/(2*sigma**2)))*(1/(np.sqrt(2*np.pi)*sigma))

def p_error(lim_inf,lim_sup,mu,sigma): # Probabilidad de error integrando pdf
    gaussiana
# el lim inferior es la dist mahalanobis/2
```

```
22      return quad(gaussiana,lim_inf,lim_sup,args=(mu,sigma))
23
24  # Estimacion ------ PARZEN ----------------------------------------------------
25  def ventana(x,h):
26  # Ventana rectangular que cumple pdf
27  # Se puede verificar la integral con: quad(ventana,-np.inf,np.inf,args=(h))
28      if np.abs(x) <= np.abs(h/2):
29          return 1/h
30      else:
31          return 0
32
33  def window(x):
34  # Ventana rectangular de altura 1 y ancho 1
35      if np.abs(x) <= np.abs(1/2):
36          return 1
37      else:
38          return 0
39
40  def window_gauss(x,h,desvio):
41  # Ventana gaussiana de media 0 y varianza sigma
42  # La media es 0 porque se centra la ventana sobre la muestra de entrenamiento
43  # Una gaussiana en 3sigma cae a casi 0, se podria usar 4sigma tambien
44  # Necesito que la gaussiana este dentro de la ventana de long h, por eso
45  # 4*sigma = h/2 => sigma = h/8
46      #sigma = h/6
47      sigma = h/(2*desvio)
48      if np.abs(x) <= np.abs(1/2):
49          return gaussiana(x,0,sigma)
50      else:
51          return 0
52
53  def select_window(gauss,x,h,desvio):
54  # funcion para seleccionar la ventana a utilizar: gaussiana o rectangular
55      if gauss == 1:
56          return window_gauss(x,h,desvio)
57      else:
58          return window(x)
59
60  def estimacion_parzen(muestras,soporte,h,gauss,desvio):
61  # pdf estimada en 1d por ventana de parzen
62      N = len(muestras)
63      p_hat = np.zeros(len(soporte))
64      for x in range(len(soporte)):
65          for x_i in muestras:
66              #p_hat[x] = p_hat[x] + window((soporte[x]-x_i)/h)/h
67              p_hat[x] = p_hat[x] + select_window( gauss, (soporte[x]-x_i)/h ,
68                                                   h, desvio)/h
69          p_hat[x] = p_hat[x]/N
70      return p_hat
71
72  # Estimacion ------ Kn vecinos ------------------------------------------------
73  def vol_k_vecinos(muestras,k,x):
74  # Obtengo los k vecinos mas cercanos a x
75      distancia = abs(muestras - x) # obtengo las distancias entre las muestras
76                                    # y el valor donde estoy
77      distancia_ordenada = sorted(distancia) #distancias de menor a mayor
78      v = distancia_ordenada[k-1] #k vecino mas cercano esta a distancia v de x
79      return v*2
80
81  def estimacion_kn(muestras,soporte,k):
82  # se realiza la estimacion por kn vecinos
```

```
    N = len(muestras)
    p_hat = np.zeros(len(soporte))
    if k == 1:
        for x in range(len(soporte)):
            p_hat[x] = k/(2*abs(x-vol_k_vecinos(muestras,k,soporte[x])))
        return p_hat
    else:
        for x in range(len(soporte)):
            p_hat[x] = k/(N*vol_k_vecinos(muestras,k,soporte[x]))
        return p_hat


# obtencion de la region de desicion con las densidades estimadas ------------

def decision_bound(soport,p_estimada_a,pap_a,p_estimada_b,pap_b):
# analizo punto a punto en el soporte la diferencia entre discriinantes,
# luego devuelvo el x con menor diferencia como boundary x_B
# la region teorica dio aprox en  x_B = 2,
# entonces la estimada deberia estar cerca, digo que entre -5 y 10,
# como el paso es de a 0.1: -15 = 0 y 20 = 350 => indice = x*10 + 150
    x_inf = -5
    x_sup = 10
    inf = x_inf * 10 + 150
    sup = x_sup * 10 + 150
    p_estimada_a = p_estimada_a[inf:sup]
    p_estimada_b = p_estimada_b[inf:sup]
    discrim_a = np.log(p_estimada_a) + np.log(pap_a)
    discrim_b = np.log(p_estimada_b) + np.log(pap_b)

    diferencia_ab = abs(discrim_a - discrim_b)
    index_min = np.argmax(diferencia_ab)

    bound = soport[index_min]
    return bound


# Funcion que devuelve la imagen en x ----------------------------------------
def f(dominio,imagen,x):
#se modela que la imagen vale lo mismo desde dominio-delta hasta dominio+delta
    paso = dominio[1]-dominio[0]
    delta = paso/2
    for i in range(len(dominio)):
        if (x >= dominio[i]-delta and x < dominio[i]+delta):
            index = i
            return imagen[index]


# KNN Clasificacion ----------------------------------------------------------
def agregar_dist_clase(muestra,clase,x):
#Obtengo las distancias ordenadas por cercania de cada punto a la referencia x
# Devuelvo una matriz Nx2: col1=distancias, col2=clase(1 o 0)
    N = len(muestra)
    if clase == 1:
        clase = np.ones(N)
    else:
        clase = np.zeros(N)
# se obtiene la distancia en modulo de cada muestra de entrenamiento hasta la
# x a clasificar
    distancia = abs(muestra-x)
# se ordena a las muestras de entrenamiento por cercania a la x a clasificar
    dist_ordenada = sorted(distancia)
```

```
# se agrega una columna con la clase de las muestras de entrenamiento
    aux = np.column_stack((dist_ordenada,clase))
    return aux

def es_clase_a_KNN(muestra_a,muestra_b,x,k):
# obtengo array de distancias ordenadas por cercania a x y clase(0 o 1)
    dist_x_a = agregar_dist_clase(muestra_a,0,x)
    dist_x_b = agregar_dist_clase(muestra_b,1,x)
# obtengo un unico array con distancias y clases
    z = np.vstack((dist_x_a, dist_x_b))
# ordeno el array por cercania a punto x
    dist_x = np.array(sorted(z,key=lambda x: x[0]))
# me quedo con la columna de clase
    clases = dist_x[:,1]
# cuento la cantidad de k vecinos de clase = 1, es decir clase b
    suma = sum(clases[0:k])
    if suma >= np.floor(k/2): # si tengo muchos 1 es porque es de clase b
        return 1 # no es de clase a, devuelvo un 1
    else: # suma < k/2, o sea que es de clase = 0, es decir clase a
        return 0 # es de clase a y devuelvo 0

def sep_clases_KNN(muestra_a,muestra_b,array_muestra,k):
# Devuelvo dos arrays, uno con las muestras de la clase a
# y otro de las de clase b por regla KNN
    array_clase_a = np.zeros(0)
    array_clase_b = np.zeros(0)
#    print(array_muestra[0])
    for i in range(len(array_muestra)):
        aux = es_clase_a_KNN(muestra_a,muestra_b,array_muestra[i],k)
        if aux == 0: #array_muestra[i] es de clase a
            array_clase_a = np.append(array_clase_a,array_muestra[i])
        else:
            array_clase_b = np.append(array_clase_b,array_muestra[i])
    return array_clase_a,array_clase_b

def sep_clases_agregar_col_KNN(muestra_a,muestra_b,array_muestra,k):
# Devuelvo array_muestra con una columna extra con la clasificacion obtenida
    array_clasif = np.zeros(0)
    for i in range(len(array_muestra)):
        clase = es_clase_a_KNN(muestra_a,muestra_b,array_muestra[i][0],k)
        array_clasif = np.append(array_clasif,clase)
    aux = np.column_stack((array_muestra,array_clasif))
    return aux


# CLASIFICADOR DICOTOMICO ----------------------------------------------------
# Con las distribuciones estimadas y la probabilidad a priori se arma un
#clasificador bayesiano dicotomico.
def es_clase_a(p_priori_a,p_estimada_a,p_priori_b,p_estimada_b,muestra,soporte):
    p_a = p_priori_a*f(soporte,p_estimada_a,muestra)
    p_b = p_priori_b*f(soporte,p_estimada_b,muestra)
    #g = p_a - p_b
    if p_a > p_b: # es de clase a
        return 0
    else: #es de clase b
        return 1

def sep_clases(p_priori_a,p_estimada_a,p_priori_b,p_estimada_b,
               array_muestra,soporte):
# Devuelvo dos arrays, uno con las muestras de la clase a y otro de la b
    array_clase_a = np.zeros(0)
```

```
205     array_clase_b = np.zeros(0)
206     for i in range(len(array_muestra)):
207         aux = es_clase_a(p_priori_a,p_estimada_a,p_priori_b,p_estimada_b,
208                          array_muestra[i],soporte)
209         if aux == 0: #array_muestra[i] es de clase a
210             array_clase_a = np.append(array_clase_a,array_muestra[i])
211         else:
212             array_clase_b = np.append(array_clase_b,array_muestra[i])
213     return array_clase_a,array_clase_b
214
215 def sep_clases_agregar_col(p_priori_a,p_estimada_a,p_priori_b,p_estimada_b,
216                            array_muestra,soporte):
217 # Devuelvo array_muestra con una columna extra con la clasificacion obtenida
218     array_clasif = np.zeros(0)
219     for i in range(len(array_muestra)):
220         clase = es_clase_a(p_priori_a,p_estimada_a,p_priori_b,p_estimada_b,
221                            array_muestra[i][0],soporte)
222         array_clasif = np.append(array_clasif,clase)
223     aux = np.column_stack((array_muestra,array_clasif))
224     return aux
225
226 def agregar_clase(array,clase):
227 # Agrego una columna extra con la clase de la muestra
228     N = len(array)
229     if clase == 1:
230         clase = np.ones(N)
231     elif clase == 0:
232         clase = np.zeros(N)
233     aux = np.column_stack((array,clase))
234     return aux
235
236
237 def cant_miss(muestra):
238 # Se obtiene la cantidad de etiquetas diferentes,
239 # es decir la cantidad de muestras clasificadas erroneamente
240     cant = 0
241     for i in range(len(muestra)):
242         if muestra[i][2] != muestra[i][1]:
243             cant = cant + 1
244     return cant
245
246 def hacer_clasificacion(imprimir, papw1, pF1, papw2, pF2,
247                         sample_F1test, sample_F2test, soport, h_k,
248                         label_estimacion,label_impresion,bound_t,bound_e):
249 # ---- Clasificacion
250 # Utilizo las muestras de prueba para clasificar con las densidades estimadas
251 # Se separan las muestras clasificandolas
252     test_class_1_F1, test_class_2_F1 = sep_clases(papw1, pF1, papw2, pF2,
253                                                   sample_F1test[:,0],soport)
254     test_class_1_F2, test_class_2_F2 = sep_clases(papw1, pF1, papw2, pF2,
255                                                   sample_F2test[:,0],soport)
256 # Se clasifican las muestras poniendole una etiqueta de clasificacion
257     muestras_1 = sep_clases_agregar_col(papw1, pF1, papw2, pF2,
258                                         sample_F1test,soport)
259     muestras_2 = sep_clases_agregar_col(papw1, pF1, papw2, pF2,
260                                         sample_F2test,soport)
261 # Se calcula la cantidad de muestras que fueron mal clasificadas
262     cant_miss_F1 = cant_miss(muestras_1)
263     cant_miss_F2 = cant_miss(muestras_2)
264 # Se obtiene erl error de clasificacion
265     err_clasif_F1 = cant_miss_F1 / len(muestras_1)
```

```
    err_clasif_F2 = cant_miss_F2 / len(muestras_2)
# Se redondea el error de clasif con 4 cifras significativas
    err_clasif_F1 = round(err_clasif_F1, 4)
    err_clasif_F2 = round(err_clasif_F2, 4)

    label_titulo = str('Clasificaci n de 100 muestras usando las distribuciones
    estimadas.\n')+label_estimacion+str(h_k)+str('.\nError de clasificaci n: Clase
     F1=')+str(err_clasif_F1)+str(', Clase F2=')+str(err_clasif_F2)

    fig_graf = graf_puntos_clasif(sample_F1test[:,0],sample_F2test[:,0],
                                  test_class_1_F1,test_class_2_F1,
                                  test_class_1_F2,test_class_2_F2,
                                  label_titulo,bound_t,bound_e)
    if imprimir == 1:
        output_filename = 'fig_d-Muestras-Clasif-' + label_impresion + '.png'
        fig_graf.savefig(output_filename,bbox_inches='tight')


# Graficos ------------------------------------------------------------------
def graf_gaussianas(soporte,p_teo_F1,p_teo_F2,bound):
# Se grafica la distribucion real y la estimada
# p_teo_F1 y p_teo_F2 son las distribuciones reales teoricas
    fig = plt.figure()
    plt.plot(soporte,p_teo_F1,color='b',linestyle='solid',linewidth=1,
             label='F1 te rica: N(1,4)')
    plt.plot(soporte,p_teo_F2,color='r',linestyle='solid',linewidth=1,
             label='F2 te rica: N(4,4)')
    plt.axvline(x=bound, color='k', linestyle='dashed',linewidth=1.2,
                label = 'Regi n de desici n, x_B='+str(round(bound,6)))
    plt.xlabel('Soporte')
    plt.ylabel('Distribuci n')
    plt.title('Distribuciones te ricas y regi n de desici n.')
    plt.grid()
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.07), ncol=2)
    plt.show()
    return fig

def graf_histograma(muestras_1,label_1,muestras_2,label_2,n,mu_1,mu_2):
# Se grafica el histograma standard de las muestras generaddas
    fig = plt.figure()
    plt.hist(muestras_1,bins='auto',alpha = 0.65,label='F1:'+label_1)
    plt.hist(muestras_2,bins='auto',alpha = 0.65,label='F2:'+label_2)
    #plt.axvline(x=mu_1, color='k', linestyle='dashed',linewidth=0.5)
    #plt.axvline(x=mu_2, color='k', linestyle='dashed',linewidth=0.5)
    plt.xlabel('Soporte')
    plt.ylabel('Cantidad de ocurrencias')
    plt.title('Histograma de '+str(n)+' muestras para F1 y F2.')
    plt.legend()
    plt.show
    return fig

def graf(soporte,p_F1,p_F2,p_teo_F1,p_teo_F2,label,h_k):
# Se grafica la distribucion real y la estimada
# p_teo_F1 y p_teo_F2 son las distribuciones reales teoricas
# p_F1 y p_F2 son las distribuciones estimadas
# h_k es el valor de longitud de ventana h o cant de vecinos kn usados
    fig = plt.figure()
    plt.plot(soporte,p_F1,color='b',linestyle='dashed',linewidth=1.2,
             label='F1 estimada')
    plt.plot(soporte,p_F2,color='r',linestyle='dashed',linewidth=1.2,
             label='F2 estimada')
```

```
325     plt.plot(soporte,p_teo_F1,color='b',linestyle='solid',linewidth=1,
326              label='F1 te rica')
327     plt.plot(soporte,p_teo_F2,color='r',linestyle='solid',linewidth=1,
328              label='F2 te rica')
329     plt.xlabel('Soporte')
330     plt.ylabel('pdf estimada')
331     plt.title('Comparaci n entre distribuci n te rica y estimada.\n'
332               +label+str(h_k))
333     plt.grid()
334     plt.legend()
335     plt.show()
336     return fig
337 
338 def graf_puntos_clasif(real_F1,real_F2,clas1_F1,clas2_F1,clas1_F2,clas2_F2,
339                        label_title,bound_t,bound_e):
340 # Se grafican las muestras a clasificar y clasificadas
341 # real_F1 y real_F2 son las muestras a clasificar provenientes de F1 y F2
342 # Clas1_F1 son las muestras de F1 clasificadas como F1
343 # Clas2_F1 son las muestras de F1 clasificadas como F2
344 # Clas1_F2 son las muestras de F2 clasificadas como F1
345 # Clas2_F2 son las muestras de F2 clasificadas como F2
346     fig = plt.figure()
347 # Boundarteorica y estimada
348     plt.axvline(x=bound_t, color='k', linestyle='dashed',linewidth=1,
349                 label = 'Regi n de desici n teorica, x_BT='+str(round(bound_t,4)
    ))
350     plt.axvline(x=bound_e, color='k', linestyle='dashdot',linewidth=1,
351                 label = 'Regi n de desici n estimada, x_Be='+str(round(bound_e
    ,4)))
352 # Recuadro del muestras
353     naranja = '#f5be58'
354     plt.axhspan(1.9, 2.1, alpha=0.35, color = naranja,
355                 label = 'Muestras a clasificar')
356     plt.axhspan(-0.1, 0.1, alpha=0.15, color = 'b' ,
357                 label = 'Clasificaci n de las provenientes de F1')
358     plt.axhspan(0.9, 1.1, alpha=0.15, color = 'r' ,
359                 label = 'Clasificaci n de las provenientes de F2')
360 # Datos
361     plt.plot(clas1_F1,len(clas1_F1)*[0],'bx',linewidth=1,label='clase F1')
362     plt.plot(clas2_F1,len(clas2_F1)*[0],'r.',linewidth=1,label='clase F2')
363 
364     plt.plot(real_F1,len(real_F1)*[2],'bx',linewidth=1)
365     plt.plot(clas1_F2,len(clas1_F2)*[1],'bx',linewidth=1)
366     plt.plot(clas2_F2,len(clas2_F2)*[1],'r.',linewidth=1)
367     plt.plot(real_F2,len(real_F2)*[2],'r.',linewidth=1)
368 
369     plt.title(label_title)
370     plt.grid()
371     plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.07), ncol=1)
372     plt.show()
373     return fig
374 
375 def graf_puntos_clasif_KNN(real_F1,real_F2,clas1_F1,clas2_F1,clas1_F2,clas2_F2,
376                        label_title,bound_t):
377 # Se grafican las muestras a clasificar y clasificadas
378 # real_F1 y real_F2 son las muestras a clasificar provenientes de F1 y F2
379 # Clas1_F1 son las muestras de F1 clasificadas como F1
380 # Clas2_F1 son las muestras de F1 clasificadas como F2
381 # Clas1_F2 son las muestras de F2 clasificadas como F1
382 # Clas2_F2 son las muestras de F2 clasificadas como F2
383     fig = plt.figure()
```

```
# Boundary teorico y estimado
    plt.axvline(x=bound_t, color='k', linestyle='dashed',linewidth=1,
                label = 'Regi n de desici n teorica, x_BT='+str(round(bound_t,4)
    ))
# Recuadro del muestras
    naranja = '#f5be58'
    plt.axhspan(1.9, 2.1, alpha=0.35, color = naranja,
                label = 'Muestras a clasificar')
    plt.axhspan(-0.1, 0.1, alpha=0.15, color = 'b' ,
                label = 'Clasificaci n de las provenientes de F1')
    plt.axhspan(0.9, 1.1, alpha=0.15, color = 'r' ,
                label = 'Clasificaci n de las provenientes de F2')
# Datos
    plt.plot(clas1_F1,len(clas1_F1)*[0],'bx',linewidth=1,label='clase F1')
    plt.plot(clas2_F1,len(clas2_F1)*[0],'r.',linewidth=1,label='clase F2')
    plt.plot(real_F1,len(real_F1)*[2],'bx',linewidth=1)
    plt.plot(clas1_F2,len(clas1_F2)*[1],'bx',linewidth=1)
    plt.plot(clas2_F2,len(clas2_F2)*[1],'r.',linewidth=1)
    plt.plot(real_F2,len(real_F2)*[2],'r.',linewidth=1)

    plt.title(label_title)
    plt.grid()
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.07), ncol=1)
    plt.show()
    return fig


def graficar_ventanas(h):
# Grafico las ventanas de parzen gaussiana con desvio h,h/2,h/4,h/8
# y rectuangular de long h
    soporte = np.arange(-h/2,h/2,step=0.01)
    y1 = gaussiana(soporte,0,h/1)
    y2 = gaussiana(soporte,0,h/2)
    y6 = gaussiana(soporte,0,h/6)
    y4 = gaussiana(soporte,0,h/4)
    y8 = gaussiana(soporte,0,h/8)

    fig = plt.figure()
    plt.plot(soporte, y1, linewidth=1.2,label='N(0,h^2)')
    plt.plot(soporte, y2, linewidth=1.2,label='N(0,(h/2)^2)')
    plt.plot(soporte, y4, linewidth=1.2,label='N(0,(h/4)^2)')
    plt.plot(soporte, y6, linewidth=1.2,label='N(0,(h/6)^2)')
    plt.plot(soporte, y8, linewidth=1.2,label='N(0,(h/8)^2)')
    plt.plot((-h/2,h/2), (1,1), 'k-',linewidth=1.2, label='Rectangular h')
    plt.plot((-h/2,-h/2), (0,1), 'k--',linewidth=1.2)
    plt.plot((h/2,h/2), (0,1), 'k--',linewidth=1.2)
    plt.xlabel('Soporte')
    plt.ylabel('Amplitud de ventana')
    plt.title('Comparaci n entre ventanas utilizadas, h=' + str(h))
    plt.grid()
    plt.legend()
    plt.show()
    return fig


################################################################################
######### MAIN #################################################################
################################################################################


pap_w1 = 0.4 #probabilidad a priori de la clase 1
```

```
pap_w2 = 0.6 #probabilidad a priori de la clase 2

F1_mu = 1 #Media de la distribucion 1
F1_sigma = 4 #Varianza de la distribucion 1
F2_mu = 4 #Media de la distribucion 2
F2_sigma = 4 #Varianza de la distribucion 2

# El soporte va a ser 0 para valores menores a (muestra_min - h/2) y mayores a
# (muestra_min + h/2)
# Para el ejercicio son dos gaussianas de varianza 4 y media 1 y 4, no deberia
# tener nada fuera de un soporte (-15;20)
soporte_minimo = -15
soporte_maximo = 20
#soporte_minimo =  np.floor(min(sample_F1)-h/2)
#soporte_maximo = np.ceil(max(sample_F2)+h/2)
soporte = np.arange(soporte_minimo,soporte_maximo,step=0.1)

n=10**4  #cantidad de muestras de entrenamiento
n_test=10**2  #cantidad de muestras a clasificar

# Para imprimir o exportar las figuras obtenidas: imprimir = 1, sino 0
imprimir = 0

# Grafico las ventanas dependiendo del desvio, h=1 para posterior analisis
#fig_graf = graficar_ventanas(1)
#if imprimir == 1:
#    fig_graf.savefig('fig_0-compar_ventanas.png',bbox_inches='tight')

# Etiquetas utilizadas para imprimir o titulos
label_F1 = str('Normal(1,4)')
label_F2 = str('Normal(4,4)')
label_parzen = str('Estimaci n utilizando ventanas de Parzen')
label_window_rect = str(', Rectangular h=')
label_kn = str('Estimaci n utilizando Kn vecinos m s cercanos, k= ')

# Generacion de n muestras normales de media mu y varianza sigma
#sample = np.random.normal(mu,sigma,n)
sample_F1 = np.random.normal(F1_mu,F1_sigma,n)
sample_F2 = np.random.normal(F2_mu,F2_sigma,n)

# Generacion de muestras a clasificar
sample_F1_test = np.random.normal(F1_mu,F1_sigma,40) #genero con P(w1)=0.4
sample_F2_test = np.random.normal(F2_mu,F2_sigma,60) #genero con P(w2)=0.6

sample_F1_test = agregar_clase(sample_F1_test,0) #La clase w1 -> columna de 0
sample_F2_test = agregar_clase(sample_F2_test,1) #La clase w2 -> columna de 1

bound_teo = (np.log(pap_w2)-np.log(pap_w1)+1/8+2)*(4/5)
#bound = 2.024372
fig_graf = graf_gaussianas(soporte,gaussiana(soporte,F1_mu,F1_sigma),
                           gaussiana(soporte,F2_mu,F2_sigma),bound_teo)
#if imprimir == 1:
#    fig_graf.savefig('fig_0-gaussianas_boundary.png',bbox_inches='tight')

#obtencion del error teorico
error_x_F1 = p_error(bound_teo,np.inf,F1_mu,np.sqrt(F1_sigma))
#print('p(x>x_B|w1)=',round(error_x_F1[0],6))
error_F1 = error_x_F1[0] * pap_w1
#print('p(x>x_B|w1)*P(w1)=',round(error_F1,6)) #Esto es el error de clasificar la
    muestra como w2 cuando era w1
error_x_F2 = p_error(-np.inf,bound_teo,F2_mu,np.sqrt(F2_sigma))
```

```
#print('p(x<x_B|w2)=',round(error_x_F2[0],6))
error_F2 = error_x_F2[0] * pap_w2
#print('p(x<x_B|w2)*P(w2)=',round(error_F2,6)) #Error de clasificar como w1 cuando era w2
error_total_teo = error_F1 + error_F2
print('P(error)=',round(error_total_teo,6))


fig_graf = graf_histograma(sample_F1,label_F1,sample_F2,label_F2,n,F1_mu,F2_mu)
if imprimir == 1:
    fig_graf.savefig('fig_a-Histograma.png',bbox_inches='tight')


## Se analizo la utilizacion de varios h, se opto por utilizar h entre 0.8 y 1
#h = np.array([115/np.sqrt(n),100/np.sqrt(n),80/np.sqrt(n),
#              50/np.sqrt(n),25/np.sqrt(n)]) #h=1.15,1,0.8,0.5,0.25
#h = np.array([95/np.sqrt(n), 90/np.sqrt(n), 85/np.sqrt(n)]) #h=0.95,0.9,0.85
#h = np.array([120/np.sqrt(n), 110/np.sqrt(n)]) #h=1.2,1.1
h = np.array([110/np.sqrt(n)])

for i in range(len(h)):
# PARZEN ---- Estimacion con ventana GAUSSIANA ------ PARZEN ------- PARZEN --

##### sigma = h/6
    label_window_gauss = str(', Gaussiana(0,(h/6)^2) h=')
    label_imprimir = 'Parzen_winGauss(sigma=h6)_h=' + str(h[i])

    p_F1 = estimacion_parzen(sample_F1,soporte,h[i],1,3)
    p_F2 = estimacion_parzen(sample_F2,soporte,h[i],1,3)

    fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),
                    gaussiana(soporte,F2_mu,F2_sigma),
                    label_parzen+label_window_gauss,h[i])
    if imprimir == 1:
        output_filename = 'fig_b-' + label_imprimir + '.png'
        fig_graf.savefig(output_filename,bbox_inches='tight')

# ---- Clasificacion
    bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
    label_estimacion = label_parzen+label_window_gauss
    hacer_clasificacion(imprimir,pap_w1, p_F1, pap_w2, p_F2,
                        sample_F1_test, sample_F2_test, soporte, h[i],
                        label_estimacion, label_imprimir,bound_teo,bound_est)


## Se analizaron los resultados y se vio que lo mejor es sigma = h/6
## Entonces todo lo que sigue va comentado

##### sigma = h/2
#    label_window_gauss = str(', Gaussiana(0,(h/2)^2) h=')
#    label_imprimir = 'Parzen_winGauss(sigma=h2)_h=' + str(h[i])

#    p_F1 = estimacion_parzen(sample_F1,soporte,h[i],1,1)
#    p_F2 = estimacion_parzen(sample_F2,soporte,h[i],1,1)

#    fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),gaussiana(soporte,F2_mu,F2_sigma),label_parzen+label_window_gauss,h[i])
#    if imprimir == 1:
#        output_filename = 'fig_b-' + label_imprimir + '.png'
#        fig_graf.savefig(output_filename,bbox_inches='tight')

```

```
563 # ---- Clasificacion
564 # Utilizo las muestras de prueba para clasificar con las densidades estimadas
565 #    bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
566 #    label_estimacion = label_parzen+label_window_gauss
567 #    hacer_clasificacion(imprimir,pap_w1, p_F1, pap_w2, p_F2, sample_F1_test,
    sample_F2_test, soporte, h[i], label_estimacion, label_imprimir,bound_teo,
    bound_est)
568
569 ##### sigma = h/4
570 #    label_window_gauss = str(', Gaussiana(0,(h/4)^2) h=')
571 #    label_imprimir = 'Parzen_winGauss(sigma=h4)_h=' + str(h[i])
572
573 #    p_F1 = estimacion_parzen(sample_F1,soporte,h[i],1,2)
574 #    p_F2 = estimacion_parzen(sample_F2,soporte,h[i],1,2)
575
576 #    fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),gaussiana
    (soporte,F2_mu,F2_sigma),label_parzen+label_window_gauss,h[i])
577 #    if imprimir == 1:
578 #        output_filename = 'fig_b-' + label_imprimir + '.png'
579 #        fig_graf.savefig(output_filename,bbox_inches='tight')
580
581 # ---- Clasificacion
582 #    bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
583 #    label_estimacion = label_parzen+label_window_gauss
584 #    hacer_clasificacion(imprimir,pap_w1, p_F1, pap_w2, p_F2, sample_F1_test,
    sample_F2_test, soporte, h[i], label_estimacion, label_imprimir,bound_teo,
    bound_est)
585
586
587 ##### sigma = h/8
588 #    label_window_gauss = str(', Gaussiana(0,(h/8)^2) h=')
589 #    label_imprimir = 'Parzen_winGauss(sigma=h8)_h=' + str(h[i])
590
591 #    p_F1 = estimacion_parzen(sample_F1,soporte,h[i],1,4)
592 #    p_F2 = estimacion_parzen(sample_F2,soporte,h[i],1,4)
593
594 #    fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),gaussiana
    (soporte,F2_mu,F2_sigma),label_parzen+label_window_gauss,h[i])
595 #    if imprimir == 1:
596 #        output_filename = 'fig_b-' + label_imprimir + '.png'
597 #        fig_graf.savefig(output_filename,bbox_inches='tight')
598
599 # ---- Clasificacion
600 #    bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
601 #    label_estimacion = label_parzen+label_window_gauss
602 #    hacer_clasificacion(imprimir,pap_w1, p_F1, pap_w2, p_F2, sample_F1_test,
    sample_F2_test, soporte, h[i], label_estimacion, label_imprimir,bound_teo,
    bound_est)
603
604
605
606 # PARZEN ---- Estimacion con ventana RECTANGULAR ------- PARZEN ------- PARZEN
607     p_F1 = estimacion_parzen(sample_F1,soporte,h[i],0,2) #el 2 no importa aca
608     p_F2 = estimacion_parzen(sample_F2,soporte,h[i],0,2)
609
610     fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),
611                     gaussiana(soporte,F2_mu,F2_sigma),
612                     label_parzen+label_window_rect,h[i])
613     if imprimir == 1:
614         output_filename = 'fig_b-Parzen_winRect_h=' + str(h[i]) + '.png'
615         fig_graf.savefig(output_filename,bbox_inches='tight')
```

```
616
617 # ---- Clasificacion
618 # Utilizo las muestras de prueba para clasificar con las densidades estimadas
619     bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
620     label_estimacion = label_parzen+label_window_rect
621     label_imprimir = 'Parzen_winRect_h=' + str(h[i])
622     hacer_clasificacion(imprimir, pap_w1, p_F1, pap_w2, p_F2,
623                         sample_F1_test, sample_F2_test, soporte, h[i],
624                         label_estimacion, label_imprimir, bound_teo, bound_est)
625
626
627 # Kn ---- Estimacion con k vecinos ---- Kn ---- Kn ---- Kn ---- Kn ---- Kn ---
628
629
630 k=np.array([1,10,50,100])
631
632 for i in range(len(k)):
633     p_F1 = estimacion_kn(sample_F1,soporte,k[i])
634     p_F2 = estimacion_kn(sample_F2,soporte,k[i])
635     fig_graf = graf(soporte,p_F1,p_F2,gaussiana(soporte,F1_mu,F1_sigma),
636                     gaussiana(soporte,F2_mu,F2_sigma),label_kn,k[i])
637     if imprimir == 1:
638         output_filename = 'fig_c-KnVecinos_k=' + str(k[i]) + '.png'
639         fig_graf.savefig(output_filename,bbox_inches='tight')
640
641 # ---- Clasificacion
642 # Utilizo las muestras de prueba para clasificar con las densidades estimadas
643     bound_est = decision_bound(soporte,p_F1,pap_w1,p_F2,pap_w2)
644     label_imprimir = 'Kn_vecinos_k='+ str(k[i])
645     hacer_clasificacion(imprimir,pap_w1, p_F1, pap_w2, p_F2,
646                         sample_F1_test, sample_F2_test, soporte, k[i],
647                         label_kn, label_imprimir, bound_teo, bound_est)
648
649
650
651 # ---- Clasificacion KNN ----
652 k=np.array([1,11,51])
653
654 for i in range(len(k)):
655 # ---- Clasificacion KNN
656 # Utilizo las muestras de prueba para clasificar con las densidades estimadas
657     test_clase_1_F1, test_clase_2_F1 = sep_clases_KNN(sample_F1,sample_F2,
658                                                       sample_F1_test[:,0],k[i])
659     test_clase_1_F2, test_clase_2_F2 = sep_clases_KNN(sample_F1,sample_F2,
660                                                       sample_F2_test[:,0],k[i])
661
662     muestra_1_clasif = sep_clases_agregar_col_KNN(sample_F1,sample_F2,
663                                                   sample_F1_test,k[i])
664     muestra_2_clasif = sep_clases_agregar_col_KNN(sample_F1,sample_F2,
665                                                   sample_F2_test,k[i])
666
667     cant_miss_F1 = cant_miss(muestra_1_clasif)
668     cant_miss_F2 = cant_miss(muestra_2_clasif)
669
670     error_clasif_F1  = cant_miss_F1 / len(muestra_1_clasif)
671     error_clasif_F2  = cant_miss_F2 / len(muestra_2_clasif)
672
673     error_clasif_F1 = round(error_clasif_F1, 4)
674     error_clasif_F2 = round(error_clasif_F2, 4)
675
676     label_title = str('Clasificaci n de ')+str(n_test)+str(' muestras usando KNN,
```

```
        k=')+str(k[i])+str('.\nError de clasificaci n: Clase F1=')+str(
    error_clasif_F1)+str(', Clase F2=')+str(error_clasif_F2)
677     fig_graf = graf_puntos_clasif_KNN(sample_F1_test[:,0],sample_F2_test[:,0],
678                                       test_clase_1_F1,test_clase_2_F1,
679                                       test_clase_1_F2,test_clase_2_F2,label_title,
680                                       bound_teo)
681     if imprimir == 1:
682         output_filename = 'fig_e-Muestras-Clasif-KNN_k=' + str(k[i]) + '.png'
683         fig_graf.savefig(output_filename,bbox_inches='tight')
```