Apuntes

Reconocimiento de Patrones

José Tomás Tocino García

Junio de 2013

Índice

1. Introducción a la asignatura

El **reconocimiento de patrones** trata las técnicas de análisis y extracción de información de elementos, tanto físicos como lógicos, con objeto de establecer propiedades, clasificaciones y reglas sobre dichos objetos – esto es, identificar **patrones**, con los que llevar a cabo multitud de funciones.

1.1. Convenciones sobre los datos

Se han seguido ciertas convenciones a la hora de presentar y trabajar con los datos. Un **patrón** es un conjunto de valores relacionados que *definen* a un **individuo** de una población. Un patrón se representa con un **vector columna**, en el cada **característica** ocupa una fila. Este vector se conoce como **vector de características**.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Los **patrones** se presentarán de forma matricial con la letra X, ocupándose **una columna por patrón**.

$$X = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ x_{21} & \dots & x_{2n} \\ x_{31} & \dots & x_{3n} \end{bmatrix}$$

Por ejemplo, si estamos en una clase con 6 alumnos y a cada uno (cada *patrón*) le medimos la altura y el peso (las *características*), la matriz resultante tendría dos filas y 6 columnas:

$$X = \begin{bmatrix} 1{,}70 & 1{,}90 & 1{,}80 & 1{,}73 & 2{,}01 & 1{,}65 \\ 70 & 90 & 80 & 81 & 97 & 54 \end{bmatrix}$$

Por otro lado, tendremos una serie de **clases** entre las que dividir nuestros datos. En el ejemplo, las dos clases podrían ser **hombre** y **mujer**, por ejemplo. La clases también se representan de forma matricial, con la letra Y, que contendrá una fila (la *clase*) y tantas columnas como patrones.

Normalmente las clases se **codifican** de forma numérica, asigando (por ejemplo) el número 1 a la clase *hombre* y el 2 a la clase *mujer*:

$$Y = \begin{bmatrix} 2 & 2 & 1 & 2 & 1 & 2 \end{bmatrix}$$

2. Clasificación

Una de las principales funciones del reconocimiento de patrones es la **clasificación** de patrones utilizando la información proporcionada por otros patrones previamente clasificados de forma correcta. Se suele recibir un conjunto de patrones de entrenamiento, junto a sus clases, y un conjunto de patrones de test, para los que calcular la clase. Estudiaremos diversos métodos para la clasificación.

2.1. Diagrama de métodos de clasificación

Se listan los métodos de clasificación estudiados, según el tipo de cada uno.

- Métodos de clasificación
  - Métodos paramétricos (Para cada clase obtienen una serie de parámetros que la caracterizan)
    - Dist. mínima
      - 1 centro/clase
        - Distancia euclídea
        - Distancia manhattan
        - Distancia mahalanobis
      - N centros/clase
        - K-medias
  - Métodos no paramétricos (Usan directamente los datos de entrada para clasificar los patrones)
    - 1-NN: 1-vecino más cercano
    - K-NN: K-vecinos más cercanos

2.2. Método de la distancia mínima

El método de la **distancia mínima** clasifica los patrones según su **distancia** hacia unos puntos, conocidos como **prototipos**, que representarán las clases. Hay múltiples formas de **decidir los prototipos**, así como diferentes maneras de **calcular las distancias** hacia esos centros.

En general, el **algoritmo** de la distancia mínima sigue los siguientes pasos:

1. Calcular los centros para cada clase.
2. Calcular la distancia entre cada patrón y cada clase.
3. Asignar a cada patrón la clase cuyo centro tiene más cerca.

2.3. Selección del prototipo

Seleccionar el **prototipo** para cada clase es el primer paso del procedimiento. Al igual que los patrones, un prototipo se representa como un vector de características.

Es habitual calcular el prototipo de una clase como la **media de los patrones** pertenecientes a esa clase, fila a fila. Esto es, el valor de la característica c del prototipo será la media de los valores para c de los patrones de esa clase.

En Matlab, el cálculo del prototipo de una clase c se hace de la siguiente manera:

```
% Decidimos la clase para la que generar el prototipo
clase = 1;

% Aislamos los patrones de esa clase
x_aux = x(:, y == clase);

% Hacemos la media de las filas de esos patrones
prototipo = mean(x_aux, 2);

% O usando la biblioteca 'pattern'
prototipo = meanpat(x_aux);
```

2.4. Distancia euclídea

La forma más básica del algoritmo utiliza la **distancia euclídea** para calcular la distancia entre los patrones y los prototipos de las clases. La distancia euclídea es la distancia más habitual, ya que se ajusta a la idea intuitiva de *distancia*.

La distancia euclídea entre un patrón x y un prototipo p se calcula como:

$$d(x,p) = \sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2 + \cdots + (x_n - p_n)^2}$$

En Matlab, la distancia euclídea de un conjunto de patrones a un prototipo se puede calcular usando la función `d_euclid` de la biblioteca `pattern`:

```
% Suponemos que tenemos tres clases
for i = 1:3
  distancias(i,:) = d_euclid(x, prototipos(:, i));
end
```

O también manualmente, utilizando la función `norm`:

```
for i = 1:3
  for j = 1:size(x,2)
    distancias(i,j) = norm(x(:, j) - prototipo(:, i))
  end
end
```

Conocidas las distancias de cada punto a cada prototipo, solo queda seleccionar el prototipo más cercano a cada punto y así decidir su clase. En Matlab:

```
% Por defecto calcula los minimos por columnas
[~, pos] = min(distancias);
```

2.5. Distancia de Manhattan

La **distancia de Manhattan** recibe ese nombre porque es la habitualmente usada en ciudades con una distribución en rejilla como Manhattan. El cálculo de esta distancia es sencillo: utiliza la suma de las diferencias, en valor absoluto, de las componentes de los vectores a comparar. Así, la distancia entre un patrón x y un prototipo p se define como:

$$d(x,p) = |x_1 - p_1| + |x_2 - p_2| + \cdots + |x_n - p_n| = \sum_{i=1}^{n} |x_n - p_n|$$

El cálculo en Matlab se puede hacer de la siguiente manera:

```
for i = 1:numClases
  % Convertimos el prototipo en una matriz con columnas iguales
  aux = repmat(prototipo(:, i), 1, length(x));

  distancias(i,:) = sum(abs(x - aux))
end
```

2.6. Distancia de Mahalanobis

Existen situaciones en las que las distancias anteriores no funcionan bien, porque no tienen en cuenta más que la diferencia entre los valores de los patrones y los prototipos, pero no su rango, varianza ni ninguna otra cualidad.

Por ejemplo, supongamos un sistema en el que los patrones tienen dos características, c_1 y c_2. c_1 suele variar entre 0 y 100 y c_2 entre 10 y 15. La distancia euclídea y la de Manhattan no tienen en cuenta el rango de las características, por lo que si un patrón tiene $c_2 = 10$ y el prototipo tiene $c_2 = 15$, su distancia no será grande, aun a pesar de que la segunda característica es *totalmente* diferente.

Para solventar este problema resulta necesario incluir en el cálculo de la distancia algún otro parámetro que caracterice cada característica.

2.6.1. Matriz de covarianza

En un entorno unidimensional, la **desviación típica** (que suele representarse como σ) mide la *dispersión* de los datos respecto al valor medio. Por ejemplo, las tres muestras $(0, 0, 14, 14)$, $(0, 6, 8, 14)$ y $(6, 6, 8, 8)$ cada una tiene una media de 7. Sus desviaciones estándar son 8.08, 5.77 y 1.15 respectivamente. La tercera muestra tiene una desviación mucho menor que las otras dos porque sus valores están más cerca de la media, que es 7.

La **varianza** es el cuadrado de la desviación típica, se denota como σ^2 y es una medida alternativa de la dispersión, utilizada para facilitar ciertos cálculos.

En un entorno multidimensional, la **covarianza** mide la relación entre dos variables a la hora de crecer o decrecer. La **matriz de covarianza**, que se denota con C, es una matriz **simétrica** que guarda información sobre cómo se distribuyen las variables de un conjunto, tanto a nivel individual como a pares. En su **diagonal** se encuentra la **varianza** de cada una de las variables. El resto de celdas describe la **relación** entre cada par de variables, definidas del siguiente modo:

- Si $C(i, j) > 0$, ambas características tienden a crecer o decrecer juntas.
- Si $C(i, j) < 0$, cuando una característica crece, la otra decrece.
- Si $C(i, j) = 0$, las características son independientes entre sí.

En definitiva:

$$C = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$$

La matriz de covarianza puede calcularse en Matlab mediante el siguiente comando de la biblioteca `pattern`:

```
% Matriz de covarianza
C = covpat(x);
```

Que internamente usa la función nativa `cov`:

```
% cov recibe los patrones por filas, con las caracteristicas por columnas
C = cov(x');
```

Ejemplos de matrices de covarianza y la representación de los datos:

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

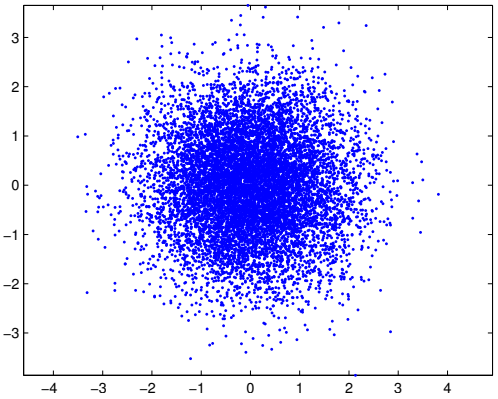


Figura 1: covarianza nula entre variables

$$C = \begin{bmatrix} 1 & 0{,}50 \\ 0{,}50 & 1 \end{bmatrix}$$

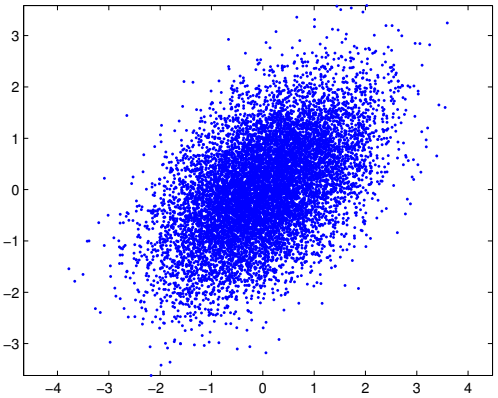


Figura 2: covarianza > 0, ambas variables crecen a la vez

$$C = \begin{bmatrix} 1 & -0{,}50 \\ -0{,}50 & 1 \end{bmatrix}$$

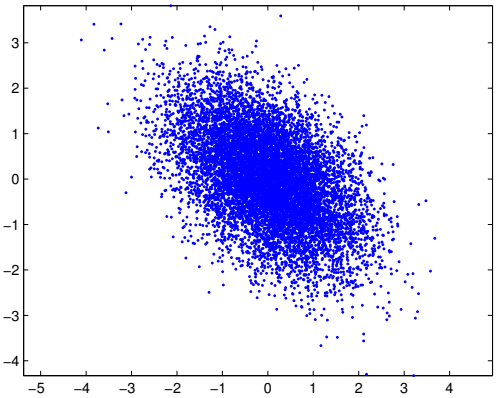


Figura 3: covarianza < 0, cuando una var. crece la otra decrece

2.6.2. Aplicación en el cálculo de la distancia

Una vez conocemos en qué consiste la matriz de covarianza, estudiaremos cómo se aplica al cálculo de la distancia. En la **distancia de Mahalanobis**, se calculará una matriz de covarianza C_i para cada clase. Hecho esto, el cálculo de la distancia de un patrón x a una clase con prototipo (media) m_i y matriz de covarianza C_i se define como:

$$d(x, m, C) = (x - m)' \cdot C^{-1} \cdot (x - m)$$

Lo que arroja un número, por el producto de matrices. En Matlab, el cálculo de la distancia de Mahalanobis de un conjunto de patrones x a una clase con media m y matriz de covarianza C se

puede hacer mediante la función `d_mahal` de la biblioteca `pattern`:

```
for i=1:numClases
  d(i,:) = d_mahal(x, m{i}, C{i});
end
```

Manualmente, podemos hacerlo de la siguiente manera:

```
for i=1:numClases
    for j=1:length(x)
       d(i,j) = ((x(:,j) - m{i})' * inv(C{i}) * (x(:, j) - m{i}));
    end
end
```

2.7. K-Medias

Existen circunstancias en las que la distribución de los puntos de cada clase es tal que el uso de un solo prototipo por clase **no es suficiente**, ya que puede darse que ese prototipo esté demasiado lejos de ciertos puntos, dando resultados falsos. En esas ocasiones es posible calcular **varios prototipos por clase**.

Las **k-medias** es un método de agrupamiento que busca dividir una clase en k grupos, perteneciendo cada patrón al grupo con la media más cercana. El algoritmo es sencillo:

1. Elegimos k centros iniciales de entre los patrones.
2. Clasificamos el resto de patrones según la cercanía a los centros.
3. Respecto a la clasificación anterior, calculamos los nuevos centros (medias) de cada clase.
4. Volvemos al punto 2.
5. La condición de parada podrá ser un número de iteraciones particular, o la convergencia.

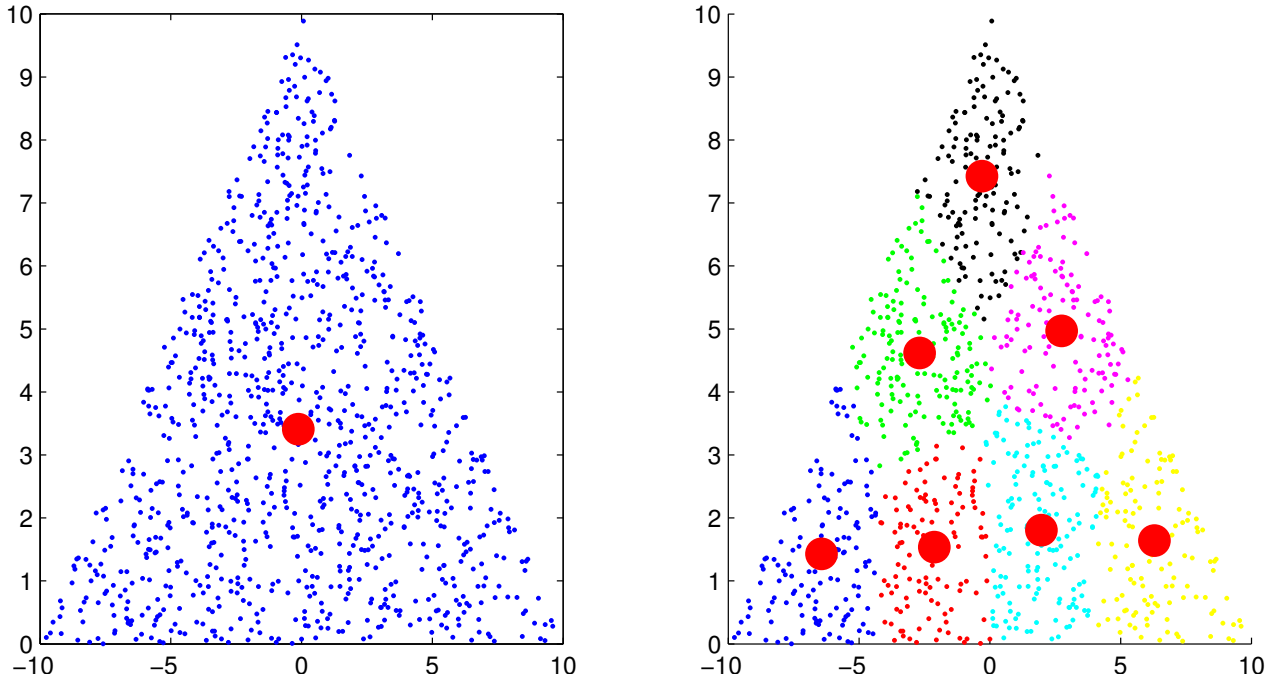


Figura 4: Ejemplo con un prototipo, y con k-medias y $k = 7$

2.7.1. Implementación

Una posible implementación del algoritmo de las k-medias en Matlab podría ser la que sigue:

```
function [ m ] = kmedias( X, numCentros )
% Algoritmo de las K-Medias para X patrones con k prototipos

% Elegimos los k centros iniciales
centros = x(:, randi(length(x), 1, k));

for i=1:50
    % Vector de distancias
    d = zeros(k, size(x, 2));

    % Calculo las distancias de cada centro a todos los patrones
    for j = 1:k
        d(j,:) = d_euclid(x, centros(:, j));
    end

    % Saco el centro mas cercano a cada patron
    [~, yaux] = min(d);

    % Genero los nuevos centros
    nuevos_centros = zeros(size(centros));
    for j=1:k
        nuevos_centros(:, j) = meanpat(x(:, yaux == j));
    end

    % Si hay convergencia, terminamos
    if isequal(centros, nuevos_centros)
        centros = nuevos_centros;
        break
    end

    centros = nuevos_centros;
end
end
```

Y una posible aplicación de esta función podría ser la siguiente, suponiendo dos poblaciones $xt1$ y $xt2$:

```
K = 5;

% Calculamos los centros
centros_x = [ kmedias(xt1, k) kmedias(xt2, k) ];
centros_y = [ ones(1, k) 2 * ones(1, k) ];

% Calculamos las distancias a los centros
for i=1:k*2
   d(i, :) = d_euclid(xtst, centros_x(:, i));
end

% Obtenemos las distancias minimas
[~, pos] = min(d);

fprintf('Errores: %d\n', sum(ytst ~= centros_y(pos)));
```

2.8. 1-NN y k-NN - Vecinos más cercanos

Hay veces en las que los patrones de las clases están muy solapados entre sí, por lo que el uso de prototipos para la clasificación no da buenos resultados. En esos casos es posible utilizar los propios patrones de entrenamiento como *representantes* de las clases. De ahí, nacen los métodos basados en los **vecinos más cercanos** (*NN - Nearest neighbors*).

Son métodos **no paramétricos**, ya que no se basan en un parámetro de la clase (como pueden ser sus centroides), sino en los propios datos de entrenamiento.

El **algoritmo** es muy sencillo. En definitiva, lo que se hace es comparar el patrón de entrada con todos los patrones de entrenamiento, teniendo en cuenta la clase del vecino más cercano (o de los k *vecinos* más cercanos) para decidir la clase del patrón de entrada. Los pasos a seguir son:

- Se reciben los patrones de entrada.
- Se calcula la distancia entre el patrón de entrada y todos los patrones de entrenamiento.
- Dependiendo del número de vecinos elegido,
  - Si se está usando 1-NN, la clase del patrón de entrada será la misma que la del vecino más cercano.
  - Si se está usando k-NN, la clase del patrón de entrada será la más frecuente entre los k vecinos más cercanos.

2.8.1. Implementación

Una posible implementación puede ser la que sigue:

```
function [ y_test ] = k_vecinos( x_train, y_train, x_test, k)

% Inicializo
y_test = zeros(1,size(x_test,2));

for i = 1:size(x_test, 2)
    % Patttern de test actual
    patron = x_test(:,i);

    % Distancia al resto de patrones de entrenamiento
    distancias = d_euclid(x_train, patron);

    % Ordeno las distancias
    [~, indices] = sort(distancias, 2);

    % Cojo las clases de los k vecinos mas cercanos
    clases_cercanas = y_train(indices(1:k));

    % Elijo la clase mas frecuente (la moda)
    y_test(i) = mode(clases_cercanas);
end

end
```

3. Cálculo del error y validación

Cuando se crea un sistema de clasificación (o cualquier otro en general) es importante tener una idea de *lo bien* que funciona el sistema, su **tasa de acierto**. A la hora de calcularla es **imprescindible** recordar que **nunca** se debe probar el sistema con los mismos datos que se han usado para entrenar el sistema, sino que **se deben usar otros**.

La forma *mala* de calcular el error (usando los datos de entrenamiento) se conoce como **error de resustitución**. Por otro lado, al calcular el error usando otros datos distintos, estaremos **estimando el error de generalización**. Es importante mencionar que es *imposible* calcular el error de generalización, ya que no es posible probar nuestro sistema con todos los posibles valores, por lo que estos métodos hacen una *estimación* de ese error.

La manera más habitual de calcular el error es utilizar la **media de los errores al cuadrado**:

$$e_i = y_i - \hat{y}(x_i), \quad e = \frac{\sum_i^n e^2}{n}$$

3.1. Validación simple

La **validación simple** es la manera más simple de dividir los datos para tener, por un lado, ciertos datos para el entrenamiento y, por otro, ciertos datos para la *validación* o **test**. Manualmente, se dividen los datos y se destina cada grupo a la tarea conveniente. También es conveniente **desordenar** los datos de entrada antes de hacer la validación, para evitar corrientes que puedan falsear el resultado.

En Matlab, el proceso habitual suele ser el siguiente, suponiendo una población inicial x de 150 elementos:

```
% Desordenamos
[x, y] = shuffle(x,y);

% 100 para entrenamiento
xtrn = x(:, 1:100);
ytrn = y(:, 1:100);

% 50 para test
xtst = x(:, 101:end);
ytst = y(:, 101:end);
```

3.2. Random sampling

Puede dar la casualidad de que la reordenación aleatoria nos de una distribución que sea poco representativa. Para evitar esos casos, podemos **repetir** el procedimiento de validación simple, de forma que en cada iteración mezclamos los datos y obtenemos el error, y al final obtenemos la media de los errores obtenidos. Este procedimiento se conoce como **error de muestreo aleatorio** o **random sampling**. La implementación es trivial:

```
for k = 1:N

    % Desordenamos los datos
    [x,y] = shuffle(x,y);

    % Datos de entrenamiento
    xtrn = x(1:12);
    ytrn = y(1:12);

    % Datos de test
    xtst = x(13:end);
    ytst = y(13:end);

    % Bucle de grado del polinomio
    for i=1:gradoMaximo

        % Calculamos los coeficientes para el grado i
        coef = polyfit(xtrn, ytrn, i);

        % Aproximamos los valores para los datos de test
        yi = polyval(coef, xtst);

        % Calculamos el error
        error(i) = error(i) + sqrt(mean((ytst-yi) .^ 2));
    end
end

% Sacamos la media del error
error = error / N;
```

3.3. Validación cruzada

Una problemática del método de muestreo aleatorio es que a menudo es muy costoso calcular el modelo tantas veces. Para esos casos, existen otras técnicas, como la de **validación cruzada** (en inglés *k-fold* cross validation). Este método divide los datos en k grupos, y en cada iteración se reserva un grupo para el test, usando el resto para entrenamiento, y calculando el error.

Así, si usamos *3-fold cross validation*, dividiremos el conjunto de datos en tres grupos, y usaremos:

train	train	**test**	1^{a} iteración, error e_1
train	**test**	train	2^{a} iteración, error e_2
test	train	train	3^{a} iteración, error e_3

La implementación de este método de validación se hace con la función `crossval` de la biblioteca `pattern`, que nos devolverá la *i-ésima* de los k posibles para esos datos.

```
[xtrn, xtst, ytrn, ytst] = crossval(x, y, num_k, i);
```

En el caso extremo en el que $k \to N$, la técnica recibe el nombre de **leave-one-out**, dado que se hacen tantas particiones como datos hay en el conjunto de entrada, y **solo se toma uno** como dato de testing.

4. Regresión

Los métodos de clasificación nos permiten asociar patrones a clases según sus propiedades. Las clases son normalmente discretas, y los errores son booleanos: el patrón estará *bien clasificado* o *mal clasificado*, no hay término medio.

Por otro lado, los **métodos de regresión** permiten aproximar valores de una distribución que a priori es desconocida, a partir de valores conocidos de esa distribución. Estos métodos intentan aproximarse a un **modelo**.

Un modelo o sistema es una **caja negra**, que recibe unas entradas x_n y tiene en su interior una función f con ciertos parámetros internos θ, que devuelve una salida y.

4.1. Modelos lineales

Un **modelo lineal** es aquél que utiliza una **combinación lineal** de los valores de entrada y sus parámetros internos para generar el valor de salida. Formalmente, dada una entrada $\vec{x} = [x_1, x_2, \ldots, x_n]$, la salida del modelo se calcula con:

$$y = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + a_0$$

Así pues, el problema de regresión para el modelo lineal se reduce a obtener el vector $\vec{A} = [a_0, a_1, \ldots, a_n]$, que son los **coeficientes** del polinomio, a partir de un conjunto de pares $(\vec{x}, y)$.

4.1.1. Demostración del cálculo de coeficientes para la recta

Vamos a demostrar cómo hacer el cálculo de los parámetros internos (los coeficientes) del modelo lineal, usando el polinomio lineal como ejemplo. El primer paso es **definir una función de error** en un punto i de la siguiente manera:

$$e_i = y_i - \hat{y}(x_i) = y_i - (a_i \cdot x + b)$$

Que es igual a la diferencia entre el valor real y el valor estimado por el sistema. Una vez definido el error en cada punto, el siguiente paso será definir *cómo unificar* los errores en todos los puntos, que es lo que se conoce como un **criterio de error**.

El criterio de error, lógicamente, dependerá de los parámetros del modelo, en este caso a y b. Además, deberá tener en cuenta que el error en cada punto puede ser positivo o negativo. La forma usual de atajar esos dos problemas es utilizando el **criterio de mínimos cuadrados**:

$$E(a,b) = \sum_{i}^{n} e^2 = \sum_{i}^{n} (y_i - (a \cdot x_i + b))^2$$

Así, el modelo *ideal* será aquél que tenga el error mínimo. El siguiente paso es **derivar** la expresión del criterio de error para conocer qué valores de a y b la minimizan.

Calculamos la derivada de E respecto de a y de b, usando: $f(x) = u^k \Longrightarrow f'(x) = k \cdot u^{k-1} \cdot u'$.

$$\frac{\partial E}{\partial a} = \sum_{i=1}^{N} 2 \cdot (y_i - (a \cdot x_i + b)) \cdot (-x_i) = 0$$
$$\frac{\partial E}{\partial b} = \sum_{i=1}^{N} 2 \cdot (y_i - (a \cdot x_i + b)) \cdot (-1) = 0$$

Ahora hay que resolver la ecuación. Operamos primero con la derivada respecto de a:

$$\sum_{i=1}^{N} \Big[2 \cdot (y_i - (a \cdot x_i + b)) \cdot (-x_i) \Big] = 0$$
$$\sum_{i=1}^{N} \Big[(2 \cdot y_i - 2(a \cdot x_i + b)) \cdot (-x_i) \Big] = 0$$
$$\sum_{i=1}^{N} \Big[2 \cdot x_i (a \cdot x_i + b) - 2 \cdot x_i \cdot y_i \Big] = 0$$
$$\sum_{i=1}^{N} \Big[2 \cdot x_i^2 \cdot a + 2 \cdot x_i \cdot b - 2 \cdot x_i \cdot y_i \Big] = 0$$
$$\sum_{i=1}^{N} \Big[2 \cdot x_i^2 \cdot a \Big] + \sum_{i=1}^{N} \Big[2 \cdot x_i \cdot b \Big] - \sum_{i=1}^{N} \Big[2 \cdot x_i \cdot y_i \Big] = 0$$
$$\sum_{i=1}^{N} \Big[x_i^2 \cdot a \Big] + \sum_{i=1}^{N} \Big[x_i \cdot b \Big] = \sum_{i=1}^{N} \Big[x_i \cdot y_i \Big]$$

Con la derivada respecto de b el procedimiento es similar

$$\sum_{i=1}^{N}\Big[2\cdot(y_i-(a\cdot x_i+b))\cdot(-1)\Big]=0$$
$$\sum_{i=1}^{N}\Big[(2\cdot y_i-2(a\cdot x_i+b))\cdot(-1)\Big]=0$$
$$\sum_{i=1}^{N}\Big[2(a\cdot x_i+b)-2\cdot y_i\Big]=0$$
$$\sum_{i=1}^{N}\Big[2\cdot x_i\cdot a+2\cdot b-2\cdot y_i\Big]=0$$
$$\sum_{i=1}^{N}\Big[2\cdot x_i\cdot a\Big]+\sum_{i=1}^{N}\Big[2\cdot b\Big]-\sum_{i=1}^{N}\Big[2\cdot y_i\Big]=0$$
$$\sum_{i=1}^{N}\Big[x_i\cdot a\Big]+\sum_{i=1}^{N}\Big[b\Big]=\sum_{i=1}^{N}\Big[y_i\Big]$$
$$\sum_{i=1}^{N}\Big[x_i\cdot a\Big]+N\cdot b=\sum_{i=1}^{N}\Big[y_i\Big]$$

Con esto, ya podemos disponer el problema como un sistema de ecuaciones de dos incógnitas en forma matricial, $A\cdot\vec{x}=B$, usando los coeficientes indicados:

$$\begin{bmatrix}\sum x_i^2 & \sum x_i\\ \sum x_i & N\end{bmatrix}\cdot\begin{bmatrix}x\\ y\end{bmatrix}=\begin{bmatrix}\sum x_i\cdot y_i\\ \sum y_i\end{bmatrix}$$

Hecho esto, podemos usar la inversa de la matriz para resolver el problema: $x=A^{-1}\cdot b$. En Matlab, podemos hacer un ejemplo completo:

```
x = [1 2 4 4 5 6 7 8];
y = [1 4 3 5 7 6 6 8];

% Construimos la matriz A con los coeficientes necesarios
n = length(x);
Sxx = sum(x.*x); Sx = sum(x);
Sxy = sum(x.*y); Sy = sum(y);
A = [Sxx Sx; Sx n]; b = [Sxy ; Sy];

% Calculamos la solucion mediante la inversa de A
sol = inv(A) * b;
% En sol tendremos los valores de a y b, que definen la recta de regresion
```

4.1.2. Generalización de la demostración

Es habitual que los modelos que usemos tengan más de una entrada, por lo que θ estará compuesto de muchos valores. La demostración en este caso es similar al caso anterior. Definimos el error en cada punto para m parámetros.

$$e_i = y_i - \hat{y}(\vec{x_i})$$

Podemos colocar el cálculo del error de forma matricial para todos los vectores de entrada.

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} & 1 \\ x_{21} & x_{22} & \dots & x_{2m} & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ x_{n1} & x_{n2} & \dots & x_{nm} & 1 \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_0 \end{bmatrix}$$

Que simbólicamente podemos representar con:

$$r = b - A \cdot x$$

OJO: a partir de aquí, la **nomenclatura** cambia:

- b es un vector fila con los valores de salida originales.
- A es una matriz con los vectores de entrada originales, más una columna de unos para el término independiente..
- x es un vector fila con los coeficientes, que son las incógnitas ae calcular.

Conociendo la definición del error en un punto, el siguiente paso es establecer un *criterio de error*, como la suma de los errores al cuadrado.

$$E = e_1^2 + e_2^2 + \dots + e_n^2 = \begin{bmatrix} e_1 & e_2 & \dots & e_n \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = r^T \cdot r$$

Basándonos en la anterior igualdad, operamos:

$$\begin{gathered} r^t r = \\ (b - Ax)^t (b - Ax) = \\ (b^t - x^t A^t)(b - Ax) = \\ b^t b - b^t AX - x^t A^t b + x^t A^t Ax \end{gathered}$$

Dado que $b^t Ax$ es de dimensiones 1×1 se cumple que

$$b^t Ax = (b^t Ax)^t = x^t A^t b$$

Por lo que la expresión previa es igual a:

$$b^tb - \overbrace{(b^tAx)^t}^{\text{lo mismo}} - \overbrace{x^tA^tb}^{\text{lo mismo}} + x^tA^tAx = b^tb - 2x^tA^tb + x^tA^tAx$$

Usando las igualdades de la sección *Anexo: derivadas habituales*, hemos de derivar por x:

$$\begin{aligned}\frac{\partial E}{\partial x} &= \frac{\partial\left[b^tb - 2b^tAx + x^tA^tAx\right]}{\partial x} \\ &= \frac{\partial\left[b^tb\right]}{\partial x} - \frac{\partial\left[2x^tA^tb\right]}{\partial x} + \frac{\partial\left[x^tA^tAx\right]}{\partial x} \\ &= 0 - 2A^tb + (A^tA + (A^t \cdot A)^t) \cdot x \\ &= 0 - 2A^tb + (A^tA + A^t \cdot A) \cdot x \\ &= 0 - 2A^tb + 2A^tAx\end{aligned}$$

Igualamos ahora a cero y obtenemos:

$$\begin{aligned}-2A^tb + 2A^tAx &= 0 \\ 2A^tAx &= 2A^tb \\ A^tAx &= A^tb \\ x &= \underbrace{(A^t \cdot A)^{-1} \cdot A^t}_{\text{Pseudoinversa de A}} \cdot b\end{aligned}$$

Y esta es la solución al ajuste de un modelo lineal con cualquier número de entradas.

4.1.3. Ejemplos

Un ejemplo básico en Matlab:

```
% Atento a que ambas son traspuestas
x = [1 2 4 4 5 6 7 8]';
y = [1 4 3 5 7 6 6 8]';

% Matriz A: columna de X, columna de unos
A = [x ones(size(x))];

% Siguiendo la eq. general
sol = inv (A' * A) * (A' *y);

% Hacemos la aproximacion con los coeficientes calculados
y2 = A * sol;

% Calculamos la diferencia con los valores reales
r = y - y2;

% Calculamos el criterio de error
E = r'*r;
```

La expresión de la pseudoinversa viene integrada en Matlab a través de la función `pinv`, como se muestra en el siguiente ejemplo con la base de datos *iris*, en la que se usan solo las flores del tipo 1 para **aproximar** el valor de la cuarta característica utilizando la información de las otras tres características.

```
load iris

% Aislamos las flores de tipo 1
x_orig = x(:, y==1);

% Cogemos las tres primeras caracteristicas para alimentar el modelo
x = x_orig(1:3,:);

% El parametro a aproximar sera la cuarta caracteristica
y = x_orig(4,:);

% Construimos la matriz B colocando en formato columna las y
b = y';

% Construimos la matriz A, con los datos establecidos por columnas segun su
% caracteristica, mas una columna de unos
A = [x' ones(size(x,2),1)];

% Calculamos los parametros
sol = pinv(A) * b;

% Hacemos la aproximacion
new_y = sol' * A';

% Ploteo de la variable original y la calculada para ver diferencias
plot(y,'r.'), hold on, plot(new_y, 'b.')
```

La gráfica que se genera es:

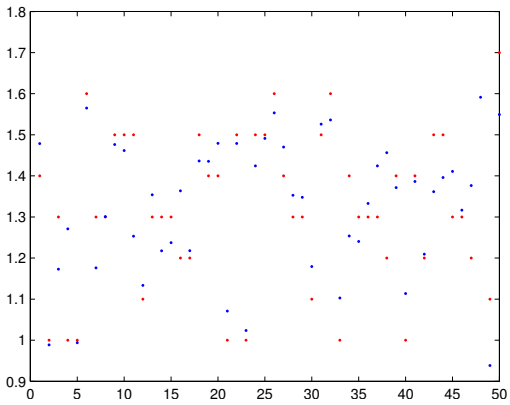


Figura 5: Comparación entre y original e y aproximada

Se ve claramente que el uso de este modelo en este caso no es apropiado, porque la tasa de error es muy alta.

4.1.4. Conversión al modelo exponencial

Existen casos en los que el fenómeno observado no se ajuste a un modelo lineal, para lo que existen otros modelos. El modelo lineal, que hemos usado hasta ahora, se ajusta a la ecuación:

$$y = b + a \cdot x$$

Por otro lado, el **modelo exponencial** se ajusta a la ecuación:

$$y = b \cdot e^{a \cdot x}$$

Es posible **convertir** el modelo lineal en el exponencial usando el logaritmo en base e:

$$y = b \cdot e^{a \cdot x} \Rightarrow \underbrace{log_e\ y}_{y^*} = \underbrace{log_e\ b}_{b^*} + a \cdot x$$

Internamente:

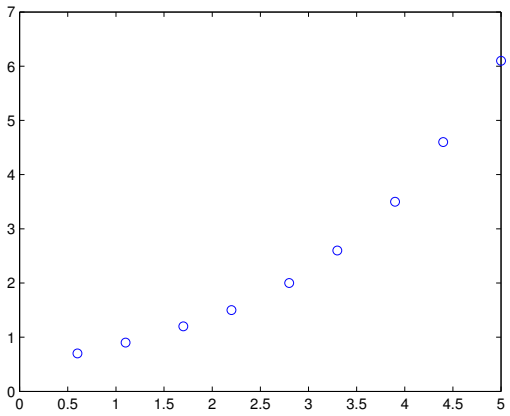
$$A = \begin{bmatrix} \uparrow & \uparrow \\ x & 1 \\ \downarrow & \downarrow \end{bmatrix}, \quad b^* = \begin{bmatrix} \uparrow \\ log_e\ y \\ \downarrow \end{bmatrix}$$

Y, tras calcular los coeficientes, deshacer el cambio de variables para obtener los coeficientes a y b^*, usando:

$$b = e^{b^*} = exp(sol(2))$$

OJO: recordar el renombramiento de variables.

Ejemplo Para el siguiente conjunto de datos:

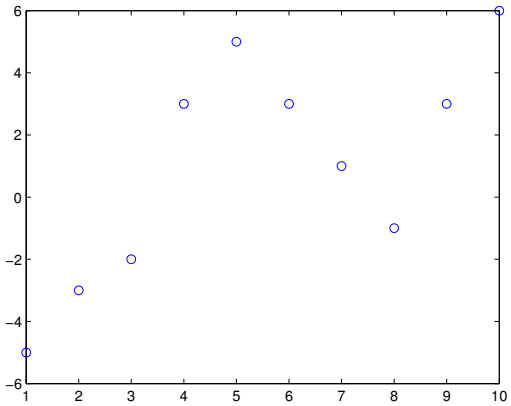


Se ve claramente que los datos no se ajustarían bien a un modelo lineal. Vamos a hacer el cálculo para un modelo exponencial.

```
% Formato columna
x=[0 0.6 1.1 1.7 2.2 2.8 3.3 3.9 4.4 5]';
y=[0.5 0.7 0.9 1.2 1.5 2 2.6 3.5 4.6 6.1]';

A = [x ones(size(x))];

% Construimos A con X y la columna de unos
A = [x ones(size(x))];

% B* es el logaritmo de y
b_estrella = log(y);

% La solucion se saca con la formula
sol = pinv(A) * b_estrella;

% El primer elemento es el primer coeficiente
a = sol(1);

% El segundo elemento es b*, por lo que elevamos e a ese elemento
b = exp(sol(2));

% Hacemos la aproximacion
ex = 0:0.1:5;
new_y = b * exp(a*ex)
```

4.1.5. Conversión al modelo polinómico

Es posible utilizar un modelo lineal para realizar un ajuste a un modelo polinómico, por ejemplo para un conjunto de datos como el siguiente:



Para ello, usaremos el truco de que dentro del modelo lineal vamos a encontrar otro modelo lineal con varias entradas. Por ejemplo, para un ajuste **cúbico**, que tiene una ecuación **similar** (que no igual) a:

$$y = a_3 \cdot x_3 + a_2 \cdot x_2 + a_1 \cdot x_1 + a_o$$

Nuestro modelo lineal tendrá la siguiente forma:

$$A = \begin{bmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ x^3 & x^2 & x & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \end{bmatrix}, \quad b^* = \begin{bmatrix} \uparrow \\ y \\ \downarrow \end{bmatrix}$$

Con esto, al resolver el modelo lineal, obtendremos los coeficientes del polinomio. En Matlab:

```
% Definimos los puntos (en forma de columnas)
x = [1:10]';
y = [-5 -3 -2  3 5 3 1 -1 3 6]';

% Construimos A y b
A = [x.^3 x.^2 x ones(size(x))];
b = y;

% Generamos los coeficientes
sol = pinv(A)*b;

% Esto coincide con el resultado de polyfit(x,y,3)
```

4.1.6. TO-DO: Conversión para la transformada de Fourier

4.1.7. Conversión a otros modelos

En general, las conversiones de otros modelos al modelo lineal son similares en tanto en cuanto siempre hay que llegar a una ecuación de la forma

$$y = b + a \cdot x$$

Ejemplo 1 Supongamos una serie de puntos $< x, y >$ que siguen una función de la forma

$$y = C \cdot x \cdot e^{A \cdot x}$$

Dado que hay una exponenciación, lo más adecuado es tomar logaritmos:

$$log(y) = log(C) + log(x) + A \cdot x$$

Reordenamos parámetros:

$$log(y) - log(x) = A \cdot x + log(C)$$

Unificamos el lado izquierdo:

$$log(\frac{y}{x}) = A \cdot x + log(C)$$

En Matlab:

```
clear all
close all
clc;

x = 1:0.1:10;
y = 4.2 .* x .* exp(2.7 .* x);

% Cambio de variable en los datos
yp = log(y'./x');

aa = [x' ones(size(x'))];
sol = pinv(aa) * yp;

% Regenero los coeficientes originales
A = sol(1);
C = exp(sol(2));

% Dibujo los puntos originales y el modelo estimado
plot (x, y, 'o'); hold on; axis auto;
xr = 0:0.01:10;
plot(xr, C.*xr.*exp((A)*xr), 'r'); hold off;
```

Ejemplo 2 Supongamos una serie de puntos $< x, y >$ que queremos ajustar a un modelo con la siguiente ecuación:

$$y = a \cdot x^b$$

Para facilitar las cosas, intercambiamos los nombres de a y b:

$$y = b \cdot x^a$$

Operamos tomando logaritmos:

$$log(y) = log(b) + log(x^a) \Rightarrow log(y) = log(b) + a \cdot log(x)$$

Una vez que tenemos la ecuación en esta forma, operamos con la pseudoinversa y deshacemos el cambio de variable.

En Matlab:

```
close all
clear all
% clc;

x = [28 50 70];
y = [75 47 34];

yp = log(y');

A = [ log(x)' ones(size(x')) ];
sol = pinv(A) * yp;

a = (sol(1));
b = exp(sol(2));

plot(x, y,'o'); hold on;
xr = 20:75;
yr = b .* xr .^ a;
plot(xr, yr, 'r'); hold off;
```

4.1.8. Anexo: derivadas habituales

Sean z y M matrices tales que:

$$z_{n\times 1} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}, \; M_{n\times k} = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1k} \\ m_{21} & m_{22} & \dots & m_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nk} \end{bmatrix}$$

Se cumplen las dos siguientes identidades:

Primera identidad

$$\frac{\partial z^t M}{\partial z} = M$$

Demostración. En efecto, tenemos que:

$$z^t_{1\times n} \cdot M_{n\times k} = P_{1\times k} = \left[z_1 \cdot m_{11} + z_2 \cdot m_{21} + \dots + z_n \cdot m_{n1} \;\; , \;\; \dots \;\; , \;\; z_1 \cdot m_{1k} + z_2 \cdot m_{2k} + \dots + z_n \cdot m_{nk} \right]$$

La derivada se expande como:

$$\frac{\partial z^t M}{\partial z} = \frac{\partial P}{\partial z} = \begin{bmatrix} \frac{\partial P}{\partial z_1} \\ \frac{\partial P}{\partial z_2} \\ \vdots \\ \frac{\partial P}{\partial z_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial P_1}{\partial z_1} & \frac{\partial P_2}{\partial z_1} & \dots & \frac{\partial P_k}{\partial z_1} \\ \frac{\partial P_1}{\partial z_2} & \frac{\partial P_2}{\partial z_2} & \dots & \frac{\partial P_k}{\partial z_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_1}{\partial z_n} & \frac{\partial P_2}{\partial z_n} & \dots & \frac{\partial P_k}{\partial z_n} \end{bmatrix}$$

Con esto, sabiendo que, por ejemplo:

$$\frac{\partial P_1}{\partial z_1} = \frac{\partial \Big[z_1 \cdot m_{11} + z_2 \cdot m_{21} + \cdots + z_n \cdot m_{n1} \Big]}{\partial z_1} = m_{11}$$

Se da pues la igualdad:

$$\frac{\partial z^t M}{\partial z} = \begin{bmatrix} \frac{\partial P_1}{\partial z_1} & \frac{\partial P_2}{\partial z_1} & \dots & \frac{\partial P_k}{\partial z_1} \\ \frac{\partial P_1}{\partial z_2} & \frac{\partial P_2}{\partial z_2} & \dots & \frac{\partial P_k}{\partial z_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_1}{\partial z_n} & \frac{\partial P_2}{\partial z_n} & \dots & \frac{\partial P_k}{\partial z_n} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1k} \\ m_{21} & m_{22} & \dots & m_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{nk} \end{bmatrix} = M$$

□

Segunda identidad

$$\frac{\partial z^t M z}{\partial z} = (M + M^t) \cdot z$$

Demostración. El producto tiene las siguientes dimensiones:

$$z^t_{1\times n} \cdot M_{n\times n} \cdot z_{n\times 1} = \square_{1\times 1}$$

Por el producto de matrices sabemos que:

$$z^t M z = \sum_{j=1}^{n} \sum_{i=1}^{n} M_{ij} \cdot z_i \cdot z_j$$

La derivada para un índice k de z sería:

$$\frac{\partial z^t M z}{\partial z_k} = \frac{\partial}{\partial z_k} \sum_{j=1}^{n} \sum_{i=1}^{n} M_{ij} \cdot z_i \cdot z_j = \sum_{i=1}^{n} M_{ik} \cdot z_i + M_{ki} \cdot z_i = \sum_{i=1}^{n} (M_{ik} + M_{ki}) \cdot z_i$$

Generalizando:

$$\frac{\partial z^t M z}{\partial z} = \begin{bmatrix} \sum_{i=1}^{n} (M_{i1} + M_{1i}) \cdot z_i \\ \sum_{i=1}^{n} (M_{i2} + M_{2i}) \cdot z_i \\ \vdots \\ \sum_{i=1}^{n} (M_{in} + M_{ni}) \cdot z_i \end{bmatrix} = \begin{bmatrix} (M_{11} + M_{11}) \cdot z_1 + (M_{21} + M_{12}) \cdot z_2 + \dots \\ (M_{12} + M_{21}) \cdot z_1 + (M_{22} + M_{22}) \cdot z_2 + \dots \\ \vdots \\ (M_{1n} + M_{n1}) \cdot z_1 + (M_{2n} + M_{n2}) \cdot z_2 + \dots \end{bmatrix} = (M + M^t) \cdot z$$

□

5. Preprocesado

5.1. Normalización

A la hora de trabajar con datos de diferentes tipos en un mismo problema, conviene **normalizar** sus valores, de forma que aunque tengan valores muy dispares, al normalizar los rangos sean similares y los diferentes métodos de cálculo, como la distancia euclídea, les den la misma importancia.

Con los datos **ordinales**, lo habitual es mapear los valores en el rango $[0, 1]$, de forma que el menor valor posible sea 0 y el mayor posible sea 1.

Con los datos **nominales** (como *blanco*, *rojo*, etc...) el problema de asignar números es que se introduce una relación implícita que es incorrecta. Por ejemplo, si a los colores les asignamos números, aquellos con números cercanos **tendrían cierta similitud** de cara al sistema, cuando realmente no es así. Para esos casos se suele calcular el número máximo N de valores discretos, y disponer N características:

blanco	1	0	0	0
rojo	0	1	0	0
azul	0	0	1	0
violeta	0	0	0	1

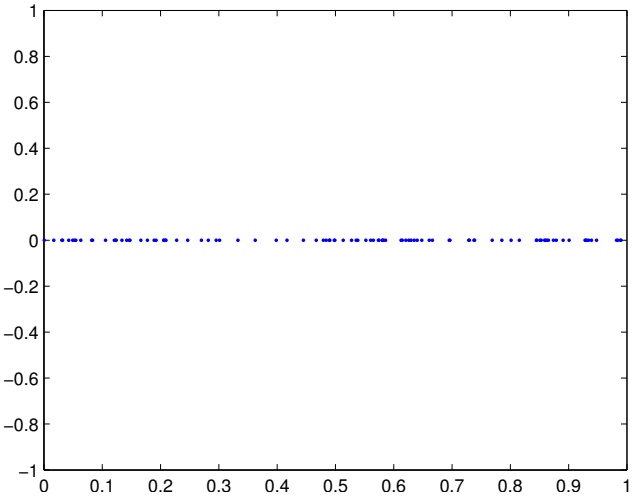
De esta forma, la distancia euclídea será la misma entre todos los valores.

Para el resto de **casos especiales** es bueno estudiarlos detenidamente de forma individualizada, para ver qué técnicas pueden ser interesantes de aplicar.

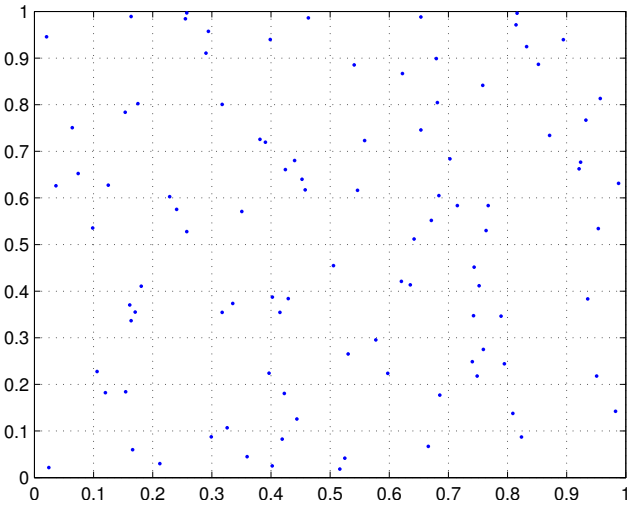
5.2. Reducción de la dimensionalidad

A medida que aumentamos el **número de características** de los patrones, va aumentando la **dimensionalidad** del problema. Es importante tener el mayor número de patrones y la mayor información posible, pero siempre dentro de una dimensionalidad adecuada.

Según aumenta la dimensionalidad, los datos van dispersándose más. Por ejemplo, supongamos el siguiente ejemplo con 100 datos en una sola dimensión:



Se ve que la densidad de datos es bastante alta en todo el dominio. Ahora, si añadimos una segunda dimensión al problema:



Empezamos a ver zonas que están vacías, que no tienen datos. Según vamos aumentando la dimensionalidad, los datos se dispersan. Es por ello **muy importante** mantener reducidas las dimensiones del problema.

5.2.1. Reducción exhaustiva

Una manera de reducir la dimensionalidad es buscar **de forma exhaustiva** las mejores características de entre las originales. Habría que comparar cada par de características, viendo cuál es más importante.

El problema es que con muchas características el proceso se hace muy **tedioso**. Una alternativa es ir añadiendo o reduciendo características y evaluando los resultados, viendo cómo influyen los cambios.

5.2.2. Reducción por combinación de características

La forma más habitual de reducir la dimensionalidad de un problema es **combinar características** de forma que, por ejemplo, un problema con 8 dimensiones se reduzca a otro de 2, evitando perder información.

Supongamos que tenemos 8 propiedades $x_1, x_2, \ldots, x_8$, y queremos reducir a 2. Definimos entonces nuevas características x' tales que:

$$\begin{aligned} x'_1 &= a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_8 \cdot x_8 \\ x'_2 &= b_1 \cdot x_1 + b_2 \cdot x_2 + \cdots + b_8 \cdot x_8 \\ &\vdots \end{aligned}$$

Con esto, para N patrones obtendremos un sistema tal que:

$$x'_{2\times N} = w_{2\times 8} \cdot x_{8\times 1000} \text{ con } w = \begin{bmatrix} a_1 & a_2 & \ldots & a_8 \\ b_1 & b_2 & \ldots & b_8 \end{bmatrix}$$

Dado que w no es una matriz cuadrada, solo sería posible recuperar una aproximación $\hat{x}$ a partir de x' y la pseudoinversa de w.

$$x \overset{w}{\Rightarrow} x' \overset{pinv(w)}{\Longrightarrow} \hat{x}$$

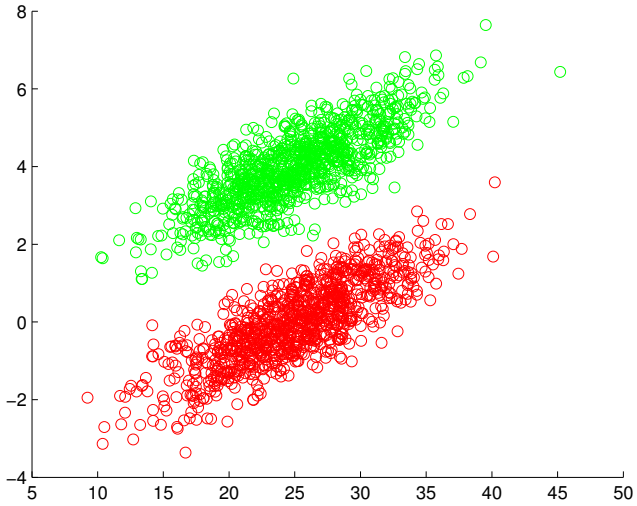
El problema, pues, reside en definir bien la matriz w tal que la diferencia entre x y $\hat{x}$ sea la menor posible.

Los dos métodos más habituales para el cálculo de w son **PCA** y **Fisher**.

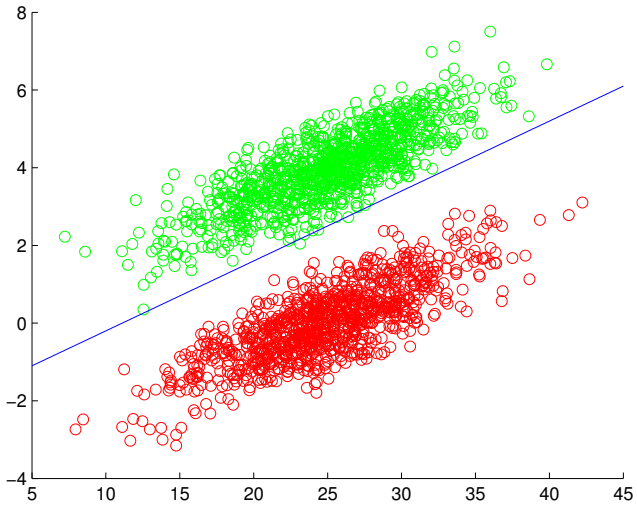
5.2.3. Reducción con PCA

PCA (*análisis de las componentes principales*) es un método clásico para la reducción de la dimensionalidad que sólo tiene en cuenta los datos de los patrones, pero no sus clases.

Supongamos que tenemos los siguientes datos en 2 dimensiones que aparecen así en la gráfica:



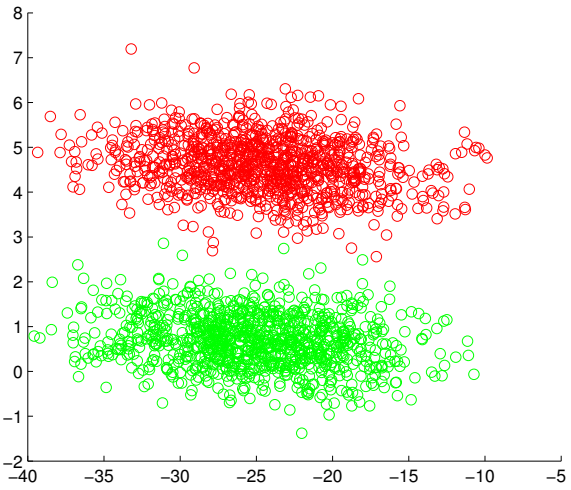
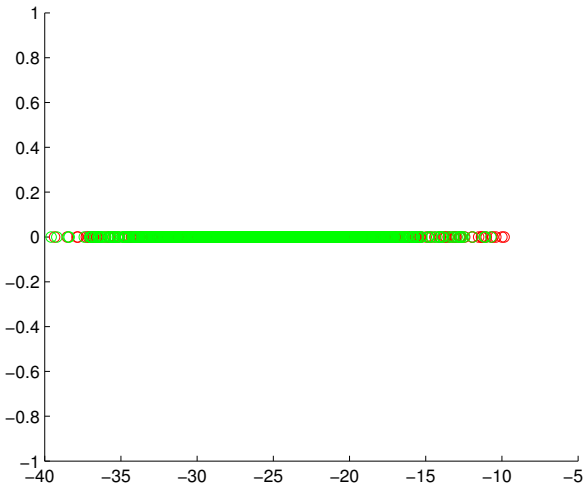
Si queremos reducir la dimensionalidad a 1, PCA buscaría la dirección en la que los datos tienen la mayor varianza, y haría la proyección, por ejemplo, sobre una línea imaginaria como la siguiente:



Lo cual, tras enderezar la proyección a 2 y 1 dimensión, quedaría:

Como vemos, se produce gran **confusión**, dado que los datos se mezclan mucho, dado que no se tiene en cuenta la clasificación.

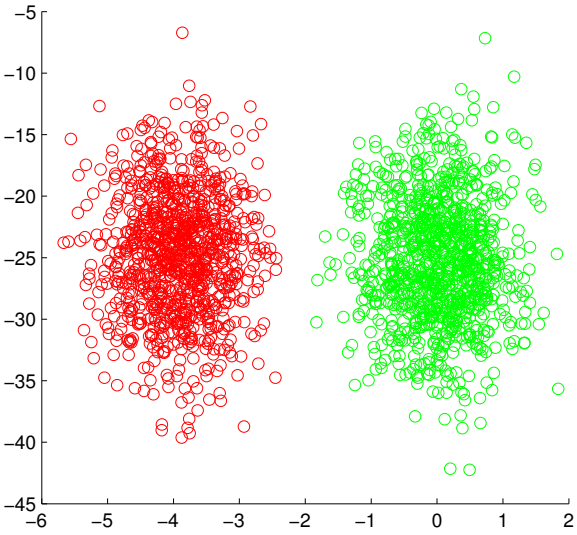
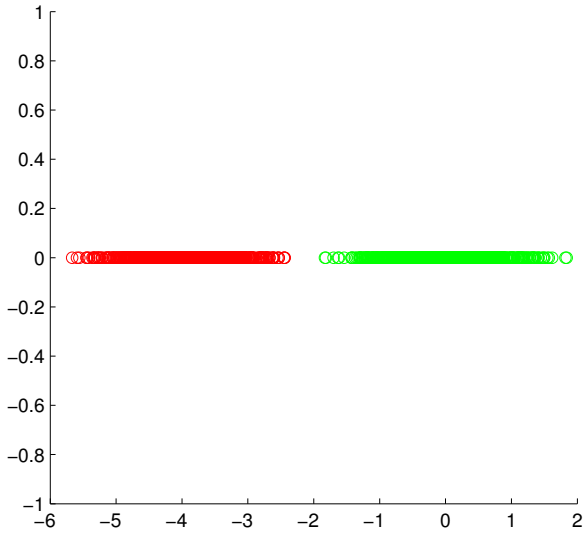
En Matlab, se haría con la función integrada `pca`.

```
W = pca(x, 1);

% Proyectamos
xprim = W * x;

% Recuperamos
xgorro = pinv(W) * xprim;
```

5.2.4. Reducción con Fisher

El método de PCA **no es adecuado para clasificación**, ya que no tiene en cuenta las clases de los patrones. En cambio, el **método de Fisher** sí es apropiado, ya que consigue que los datos de la misma categoría queden lo más cerca posible, y que las medias de esas categorías queden lo más separadas posible.

En este ejemplo, la proyección de Fisher tendría en cuenta la clasificación de los datos, y proyectaría de forma que las clases queden muy separadas.

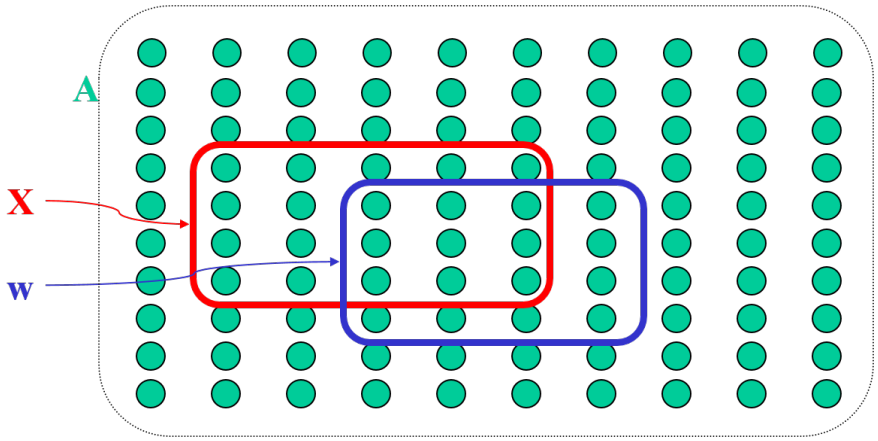
El cálculo en Matlab es similar al de PCA, solo cambiando el método.

```
% Cargamos la BD iris
load iris
y = y + 1;

% Aplicamos fisher para reducir a 2 dimensiones
W = fisher(x,y,2);

% Generamos x prima
xprim = W * x;

% Pintamos x prima
plotpat(xprim, y);
```

6. Clasificación bayesiana

6.1. Introducción

Supongamos la siguiente población de 100 elementos con la misma probabilidad $\frac{1}{100}$ de ser elegidos.



En el grupo X hay 20 elementos, por lo que la probabilidad de elegir un elemento de ese grupo de entre el total es $P(X) = \frac{20}{100} = 0{,}2$.

Por otro lado, en el grupo W hay 16 elementos, por lo que la probabilidad de elegir un elemento de W es $P(W) = \frac{16}{100} = 0{,}16$.

Ahora bien, supongamos que queremos conocer la probabilidad de la **intersección** de W con X, esto es, $w \cap X$. Intuitivamente, viendo el gráfico sabemos que esa probabilidad es igual a $\frac{9}{100}$.

Numéricamente, la probabilidad de la intersección se puede calcular de dos maneras:

1. Como la probabilidad de W multiplicada por la probabilidad de *que se dé X suponiendo que se ha dado W*, lo que se conoce como **probabilidad condicionada**. Es decir, el segundo parámetro supone que se da W, o lo que es lo mismo, que la población total es W, por lo que es igual al número de elementos de la población W que también pertenecen a X.

$$P(W \cap X) = P(W) \cdot P(X|W) = \frac{16}{100} \cdot \frac{9}{16} = \frac{9}{100}$$

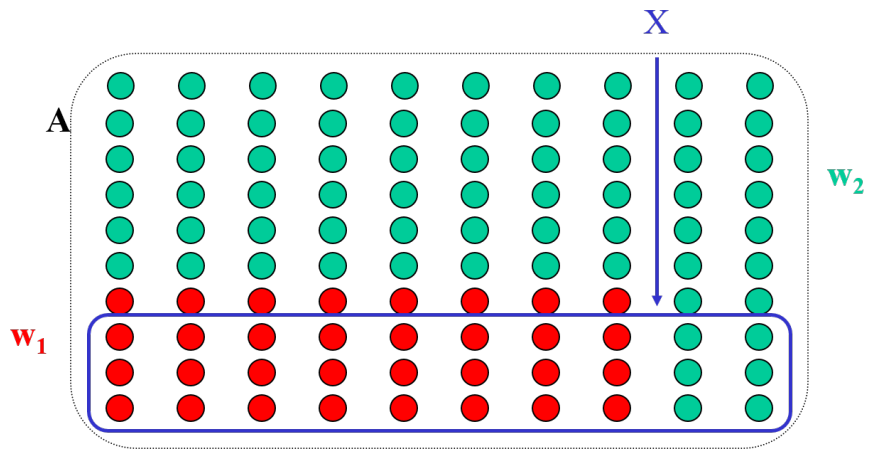
2. Como la probabilidad de X multiplicada por la probabilidad de *que se dé W suponiendo que se ha dado X*. El razonamiento es igual al caso anterior:

$$P(X \cap W) = P(X) \cdot P(W|X) = \frac{20}{100} \cdot \frac{9}{20} = \frac{9}{100}$$

6.2. Teorema de Bayes

El teorema de Bayes se obtiene operando con las anteriores igualdades, y obteniendo:

$$P(W|X) = \frac{P(X|W) \cdot P(W)}{P(X)}$$



Ejemplo Supongamos lo siguiente. De los 100 elementos totales,

- Los 32 elementos rojos son **mujeres**.
- Los 68 elementos verdes son **hombres**.
- Los 30 elementos dentro del recuadro azul (X) son alumnos **con pendiente**. Se ve que *la mayoría* de las chicas tienen pendiente, mientras que *la mayoría* de chicos **no** tienen pendiente.

Así,

$$P(w1) = \frac{32}{100},\ P(w2) = \frac{68}{100},\ P(X) = \frac{30}{100}$$

Si solo tenemos en cuenta la probabilidad de ser hombre o mujer, al elegir un sujeto aleatoriamente *lo más seguro* es que sea un hombre. Ahora bien, si del sujeto elegido sabemos una información adicional, que **tiene pendiente**, la cosa cambia, dado que intuitivamente se ve que del grupo de sujetos con pendiente *lo más seguro* es que sea una mujer.

Visualmente, vemos que las **probabilidades condicionadas** a que el sujeto tenga pendientes son:

$$P(\text{Hombre}|\text{Pendientes}) = \frac{6}{30}$$
$$P(\text{Mujer}|\text{Pendientes}) = \frac{24}{30}$$

Ahora, para fundamentar este cálculo podemos usar el teorema de Bayes.

$$\begin{aligned} P(\text{Hombre}|\text{Pendientes}) &= \frac{P(\text{Pendientes}|\text{Hombre}) \cdot P(\text{Hombre})}{P(\text{Pendientes})} \\ &= \frac{\frac{6}{68} \cdot \frac{68}{100}}{\frac{30}{100}} \\ &= \frac{6}{30} \end{aligned}$$

El cálculo para el caso de las mujeres es igual.

Nomenclatura:

$$\underbrace{P(W|X)}_{\text{Prob. a posteriori}} = \frac{\overbrace{P(X|W)}^{\text{Prob. condicional}} \cdot \overbrace{P(W)}^{\text{Prob. a priori}}}{P(X)}$$

La **nomenclatura** es muy descriptiva. Inicialmente queremos conocer si se da un hecho W, por lo que la probabilidad **a priori** que tenemos es $P(W)$.

Ahora bien, si se da una condición particular (X) y sabemos la probabilidad de que ese hecho ocurra si se da el inicial (la probabilidad **condicional**, $P(X|W)$), entonces estaremos obtenido la probabilidad **a posteriori**, es decir, la probabilidad tras conocer la circunstancia adicional.

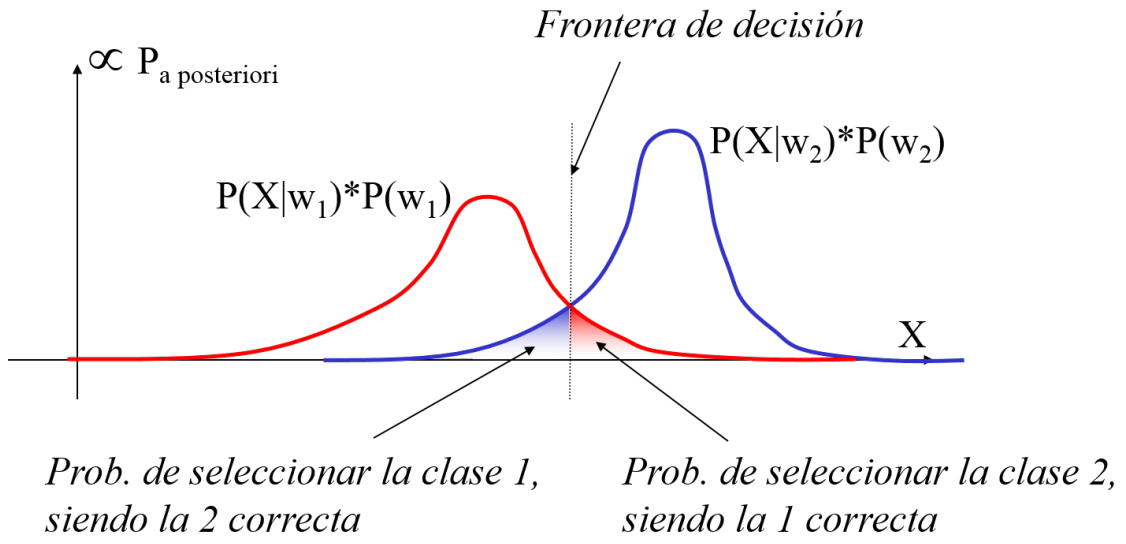
Esto tiene gran utilidad en problemas de **clasificación**, ya que conociendo ciertas propiedades (por ejemplo *tiene pendientes* y *tiene barba*), y sabiendo las probabilidades de entre la población, puedo saber la probabilidad de que sea de una clase u otra.

A la hora de seleccionar una hipótesis existen dos variantes:

- **Criterio MAP (Máximo A-Posteriori)**: consiste en elegir aquella hipótesis con la mayor probabilidad *a posteriori*. Normalmente el denominador es igual en todas las hipótesis, por lo que se puede **descartar**.
- **Criterio Máxima Verosimilitud**: consiste en comparar utilizando únicamente la **probabilidad condicional**, ya que es frecuente tratar con clases que tienen la misma probabilidad a priori.

6.3. Funciones de densidad de probabilidad

Con frecuencia, los valores con los que se trabajan son **continuos** (por ejemplo, la *altura* de un grupo de personas). En estos casos, $P(X)$ es una **función de densidad de probabilidad**. El área bajo su curva es 1.



El problema de decidir la probabilidad a posteriori se ve gráficamente como decidir la posición de la **frontera de decisión**. El primer conjunto de métodos para ello son los **métodos paramétricos**.

Suponen que las probabilidades implicadas tienen una forma dada, normalmente una **gausiana**. Para modelar las distribuciones de cada conjunto de datos simplemente se calcula la media y la desviación típica (σ^2).

```
% Calculamos la media y desv. estandar
media1 = mean(x1);
desv1 = std(x1);

% Podemos generar 1000 datos adicionales de la normal
datos_adicionales = media1 + desv1 * randn(1, 1000);

% Forma alternativa
datos_adicionales = randnorm(media1, desv1, 1000);

% Forma alternativa indicando el eje de abcisas
X = [0:0.01:30];
Y = normpdf(X, media1, desv1);
```

En una gausiana, los valores **se distribuyen así**:

$\mu \pm \sigma$	$\mu \pm 2 \cdot \sigma$	$\mu \pm 3 \cdot \sigma$
68.27 %	95.45 %	99.73 %

En este caso, la frontera de decisión óptima siempre es aquella en la que las gausianas cortan, esto es, aquella en la que las gausianas tienen el mismo valor. Nos basaremos en la definición de la función de densidad de probabilidad gausiana:

$$\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

6.3.1. Cálculo de la frontera con iguales probs. a priori e igual desv. típica

Suponiendo que las probabilidades a priori son las mismas, el cálculo de la probabilidad a posteriori se reduce a la probabilidad condicional. Suponemos también que la desviación típica es igual en ambas distribuciones ($\sigma_1 = \sigma_2$).

Igualamos:

$$\frac{1}{\sqrt{2\pi\sigma_1^2}}\, e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} = \frac{1}{\sqrt{2\pi\sigma_2^2}}\, e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}$$

Como hemos supuesto que $\sigma_1 = \sigma_2 \neq 0$, el coeficiente que multiplica puede eliminarse de ambos lados:

$$e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} = e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}$$

Al tener dos exponenciales, podemos tomar logaritmos y nos quedamos con los exponentes:

$$\frac{(x-\mu_1)^2}{2\cdot\sigma_1^2} = \frac{(x-\mu_2)^2}{2\cdot\sigma_2^2}$$

Igual que antes, factorizamos y eliminamos el coeficiente de σ:

$$(x-\mu_1)^2 = (x-\mu_2)^2$$

Expandimos los binomios al cuadrado

$$\begin{aligned} x^2 + \mu_1^2 - 2\cdot x\cdot \mu_1 &= x^2 + \mu_2^2 - 2\cdot x\cdot \mu_2 \\ \mu_1^2 - 2\cdot x\cdot \mu_1 &= \mu_2^2 - 2\cdot x\cdot \mu_2 \\ \mu_1^2 - \mu_2^2 &= 2\cdot x\cdot(\mu_1-\mu_2) \end{aligned}$$

Expandimos la parte izquierda por la regla $a^2 - b^2 = (a+b)(a-b)$:

$$\begin{aligned} (\mu_1+\mu_2)(\mu_1-\mu_2) &= 2\cdot x\cdot(\mu_1-\mu_2) \\ \mu_1+\mu_2 &= 2\cdot x \\ x &= \frac{\mu_1+\mu_2}{2} \end{aligned}$$

$x = \frac{\mu_1+\mu_2}{2}$	Así, la frontera óptima estará en esa posición si se cumplen estas restricciones: 1. Ambas curvas son normales. 2. Las probabilidades *a priori* son iguales, $P(w_1) = P(w_2)$. 3. Las desviaciones típicas son iguales, $\sigma_1 = \sigma_2$.

6.3.2. Cálculo de la frontera óptima para el caso general

Para el caso general, la frontera óptima ha de cumplir que:

$$P(X|w_1) * P(w_1) = P(X|w_2) * P(w_2)$$

Sacamos logaritmos:

$$log(\frac{1}{\sqrt{2\pi\sigma_1^2}}) - \frac{(x-\mu_1)^2}{2\sigma_1^2} + log(P(w_1)) = log(\frac{1}{\sqrt{2\pi\sigma_2^2}}) - \frac{(x-\mu_2)^2}{2\sigma_2^2} + log(P(w_2))$$

Desarrollamos el primer logaritmo sabiendo que $log(\frac{a}{b}) = log(a) - log(b)$ y que $log(\sqrt{a}) = \frac{1}{2}log(a)$:

$$log(1) - \frac{1}{2}log(2\pi) - \frac{1}{2}log(\sigma_1^2) - \frac{(x-\mu_1)^2}{2\sigma_1^2} + log(P(w_1)) = log(1) - \frac{1}{2}log(2\pi) - \frac{1}{2}log(\sigma_2^2) - \frac{(x-\mu_2)^2}{2\sigma_2^2} + log(P(w_2))$$

Quitamos de ambos lados las constantes y operamos aplicando $log(a^b) = b\cdot log(a)$:

$$-log(\sigma_1) - \frac{(x-\mu_1)^2}{2\cdot\sigma_1^2} + log(P(w_1)) = -log(\sigma_2^2) - \frac{(x-\mu_2)^2}{2\cdot\sigma_2^2} + log(P(w_2))$$

Ahora, para quitar las fracciones sacamos común denominador $2 \cdot \sigma_1^2 \cdot \sigma_2^2$:

$$\begin{gathered} -2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_1) - \sigma_2^2 \cdot ((x-\mu_1)^2) + 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_1)) \\ = \\ -2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_2) - \sigma_1^2 \cdot ((x-\mu_2)^2) + 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_2)) \end{gathered}$$

Expandimos el binomio central y movemos todo al mismo lado:

$$\begin{gathered} -2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_1) - \sigma_2^2 \cdot x^2 - \sigma_2^2 \cdot \mu_1^2 + 2 \cdot x \cdot \mu_1 \cdot \sigma_2^2 + 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_1)) \\ +2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_2) + \sigma_1^2 \cdot x^2 - 2 \cdot x \cdot \mu_2 \cdot \sigma_1^2 - 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_2)) \\ = 0 \end{gathered}$$

Ahora, agrupamos según multipliquen a x^2, a x o sean términos independientes:

$$\begin{gathered} x^2 \cdot (\sigma_1^2 - \sigma_2^2) + \\ x \cdot (2 \cdot \mu_1 \cdot \sigma_2^2 - 2 \cdot \mu_2 \cdot \sigma_1^2) \\ -2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_1) - \sigma_2^2 \cdot \mu_1^2 + 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_1)) \\ +2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(\sigma_2) + \sigma_1^2 \cdot \mu_2^2 - 2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot log(P(w_2)) \end{gathered}$$

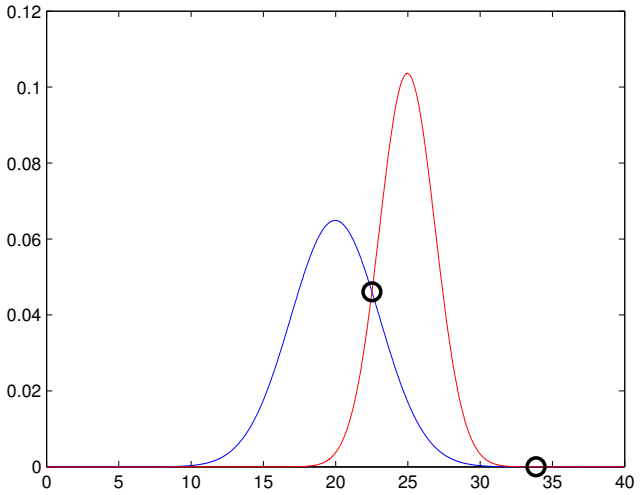
Factorizamos los términos independientes:

$$\begin{gathered} x^2 \cdot (\sigma_1^2 - \sigma_2^2) + \\ x \cdot (2 \cdot \mu_1 \cdot \sigma_2^2 - 2 \cdot \mu_2 \cdot \sigma_1^2) \\ +2 \cdot \sigma_1^2 \cdot \sigma_2^2 \cdot \Big[log(P(w_1)) - log(\sigma_1) - log(P(w_2)) + log(\sigma_2) \Big] \\ +(\sigma_1^2 \cdot \mu_2^2 - \sigma_2^2 \cdot \mu_1^2) \end{gathered}$$

Así, llegamos a una ecuación del tipo $Ax^2 + Bx + C = 0$ donde:

$$\begin{aligned} A &= \sigma_1^2 - \sigma_2^2 \\ B &= 2(\mu_1 \cdot \sigma_2^2 - \mu_2 \cdot \sigma_1^2) \\ C &= \Big[ln(P(w_1)) - ln(\sigma_1) - ln(P(w_2)) + ln(\sigma_2) \Big] \cdot 2 \cdot \sigma_1^2 \cdot \sigma_2^2 + (\sigma_1^2 \cdot \mu_2^2 - \sigma_2^2 \cdot \mu_1^2) \end{aligned}$$

Ejemplo En Matlab, suponiendo un conjunto de patrones $X_{1\times 5000}$ y $Y_{1\times 5000}$:




```
i1 = find(y == 0);
i2 = find(y == 1);

m1 = mean(x(i1));
m2 = mean(x(i2));

s1 = std(x(i1));
s2 = std(x(i2));

Pw1 = length(i1) / length(y);
Pw2 = length(i2) / length(y);

A=s1*s1-s2*s2;
B=2*(m1*s2*s2-m2*s1*s1);
C=2*s1*s1*s2*s2*(log(Pw1)-log(Pw2)-log(s1)+log(s2))+s1*s1*m2*m2-s2*s2*m1*m1;
x1=(-B+sqrt(B*B-4*A*C))/2/A
x2=(-B-sqrt(B*B-4*A*C))/2/A

I=0:0.01:40;
plot(I,Pw1*normpdf(I,m1,s1));     hold on;
plot(I,Pw2*normpdf(I,m2,s2),'r');

plot(x1, Pw1 * normpdf(x1,m1,s1), 'ko', 'LineWidth', 2, 'MarkerSize', 10);
plot(x2, Pw1 * normpdf(x2,m1,s1), 'ko', 'LineWidth', 2, 'MarkerSize', 10);

hold off;
```

6.4. Distribuciones bidimensionales

Es posible trabajar en dos dimensiones, los patrones serán vectores columnas con la forma $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, por lo que hay que adaptar la fórmula de la función de densidad de probabilidad:

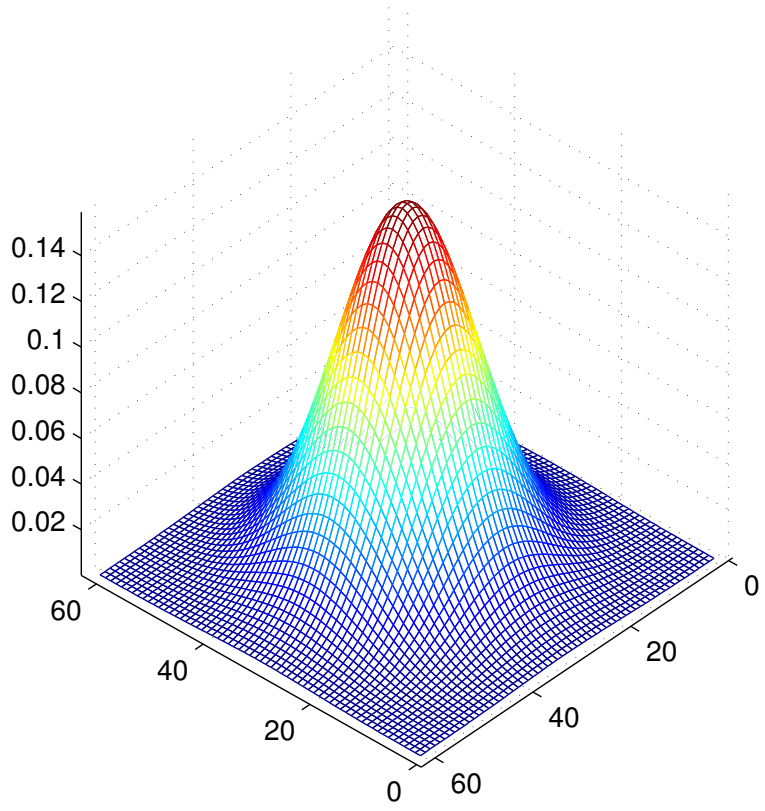
$$f_X(x_1, \ldots, x_n) = \frac{1}{(2\pi)^{n/2} \left|\Sigma\right|^{1/2}} \; e^{\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)}$$

Donde $\sum$ es la matriz de covarianza. En el caso de dos dimensiones, $n = 2$ y

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \sum = \begin{bmatrix} \sum_{11} & \sum_{12} \\ \sum_{21} & \sum_{22} \end{bmatrix}$$

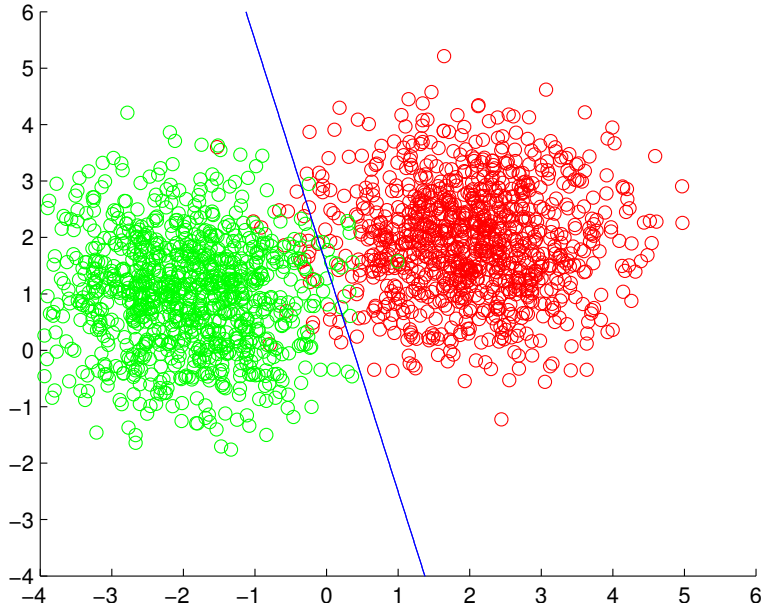
En Matlab, podemos generar una normal multivariada con mvnpdf (*multivariate normal probability density function*):

```
[x,y]=meshgrid(-3:0.1:5);

m1 = [0;0];
C1 = [1 0;0 1];
D1 = mvnpdf([x(:) y(:)], m1',C1);
D1 = reshape(D1,size(x));
mesh(D1), axis vis3d;
```



También es posible generar los valores con `randnorm`. Si hacemos un ploteo en 2 dimensiones es posible pintar la frontera con `plotbon`. La intersección de dos gausianas en 2D sigue la forma de una cónica, cuya función es:

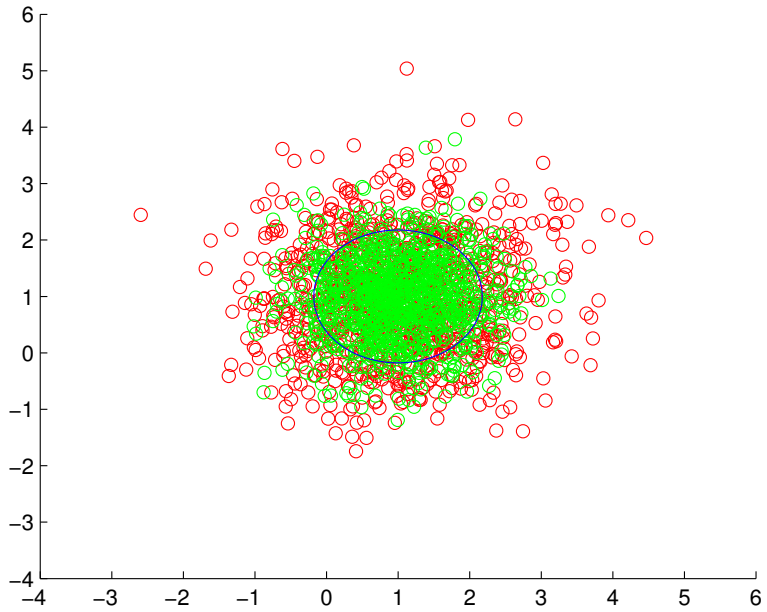
$$f(x, y) = ax^2 + by^2 + cxy + dx + ey + f$$



6.4.1. Ejemplo, medias diferentes e igual covarianza

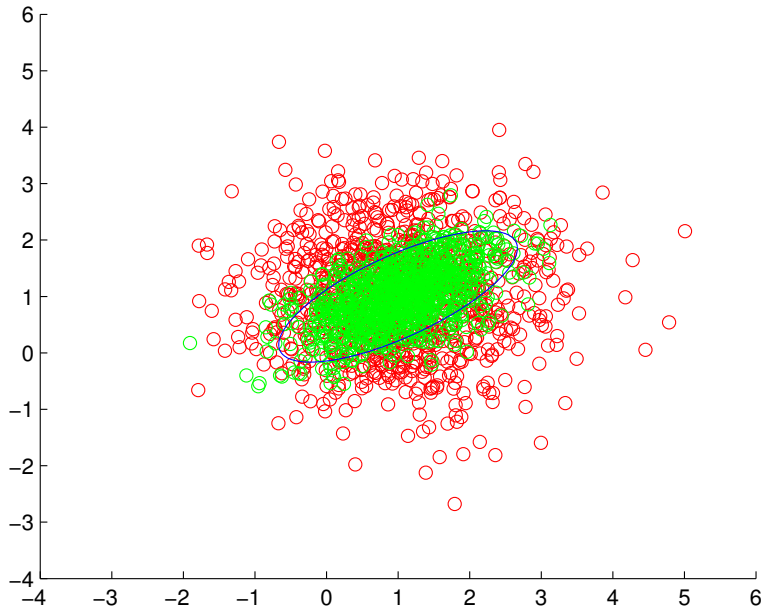
```
m1=[2;2]; m2=[-2;1]; C1=[1 0;0 1]; C2=[1 0;0 1];
x=[randnorm(m1,C1,1000) randnorm(m2,C2,1000)];
y=[zeros(1,1000) ones(1,1000)];
plotpat(x,y);
hold on;
plotbon(m1,C1,m2,C2,'b');
axis([-4 6 -4 6])
```

6.4.2. Medias iguales y covarianzas proporcionales



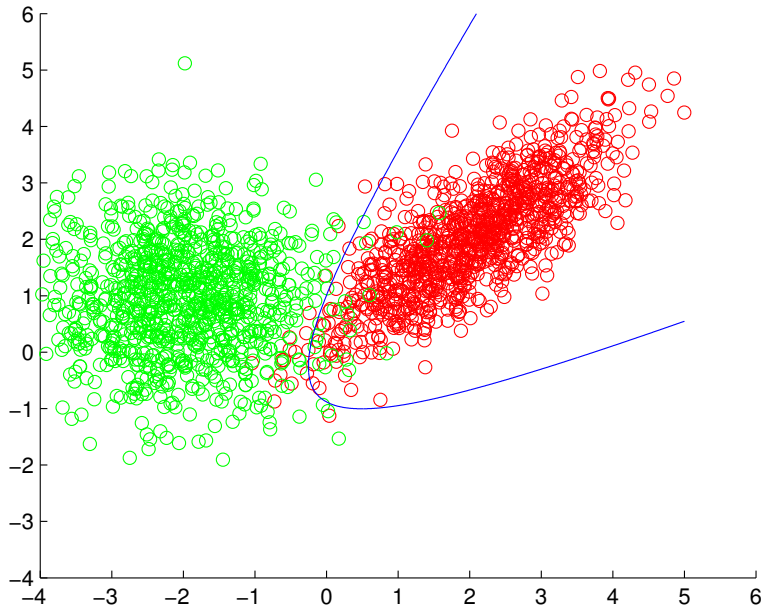

```
m1=[1;1]; m2=[1;1];
C1=[1 0;0 1]; C2=[0.5 0;0 0.5];
```

6.4.3. Medias iguales y covarianzas diferentes



```
m1=[1;1]; m2=[1;1];
C1=[1 0;0 1]; C2=[0.5 0.2;0.2 0.3];
```

6.4.4. Medias y covarianzas distintas



```
m1=[2;2]; m2=[-2;1];
C1=[1 0.8;0.8 1]; C2=[1 0;0 1];
```

6.5. Toma de decisiones con costes

Hay ocasiones en las que cometer un error en la clasificación **es más costoso** si un elemento de una clase A se coloca en la clase B que en el caso contrario.

Por ejemplo, tenemos un sistema de defensa contra misiles que detecta si un misil ha sido lanzado contra el país y lo destruye con unas contramedidas. Si el sistema que detecta la amenaza **clasifica de forma errónea** un misil real, éste acaba impactando y se producen muchas bajas, por lo que el coste del error de clasificación es **alto**.

En cambio, si da un falso positivo y cree que una gaviota es un misil, entonces el único gasto que se produce es en las contramedidas utilizadas... y en la vida de la gaviota. Por tanto, el coste del error es **bajo**.

Así, es preferible que el sistema sea más sensible a las amenazas, por el coste de un error u otro.

En estos casos se utiliza una **matriz de costes**, en la que se resume el coste de los aciertos y los errores: el elemento L_{ij} es el coste de elegir la clase j cuando la clase real es la i.

		Predicción	
	L_{ij}	1	2
Real	1	10^2	10^4
	2	10^6	10^4

6.5.1. Riesgo de predicción

Es posible estimar la pérdida general por errores de clasificación utilizando el teorema de Bayes, lo cual da lugar a la expresión del **riesgo de predicción**, donde M es el número de clases existentes:

$$r_j(x) = \sum_{i=1}^{M} L_{ij} \cdot P(x|w_i) \cdot P(w_i)$$

6.5.2. Aplicación gausiana

A la hora de aplicar la matriz de costes cuando trabajemos con distribuciones gausianas, tenemos dos opciones:

- Aplicarle el coste al cálculo del histograma:

$$hist \cdot L_{ij}$$

- Aplicarle el coste a la expresión de la probabilidad:

$$L_{12} \cdot P(X|w_1) \cdot P(w_1)$$

Esto hace que la frontera **se mueva** hacia la dirección en la que es más barato el error. El cálculo formal de la frontera se reduce a resolver la ecuación:

$$P(X|w_1) \cdot P(w_1) \cdot L_{12} = P(X|w_2) \cdot P(w_2) \cdot L_{21}$$

La demostración es básicamente igual a la que se expone en la sección *Cálculo de la frontera óptima para el caso general*, con la única diferencia de que los dos nuevos parámetros pasan al final a formar parte del término independiente del polinomio resultante, quedando así los coeficientes:

$$A = \sigma_1^2 - \sigma_2^2$$
$$B = 2(\mu_1 \cdot \sigma_2^2 - \mu_2 \cdot \sigma_1^2)$$
$$C = \Big[ln(L1) - ln(L2) + ln(P(w_1)) - ln(\sigma_1) - ln(P(w_2)) + ln(\sigma_2)\Big] \cdot 2 \cdot \sigma_1^2 \cdot \sigma_2^2 + (\sigma_1^2 \cdot \mu_2^2 - \sigma_2^2 \cdot \mu_1^2)$$

6.6. Ventanas de Parzen

Hay ocasiones en las que la distribución subyacente de los datos no es una normal, por lo que los métodos que conocemos no funcionan bien. El método de las **ventanas de Parzen** generan una distribución conjunta de muchas distribuciones individuales **centradas en cada dato**, que se suman y normalizan a área 1 dividiendo entre el número de datos. Habitualmente se utiliza con **gausianas**.

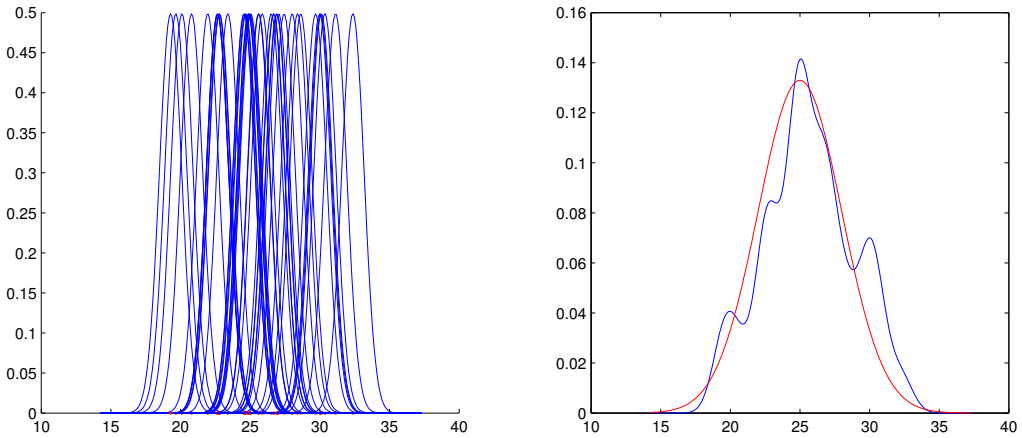


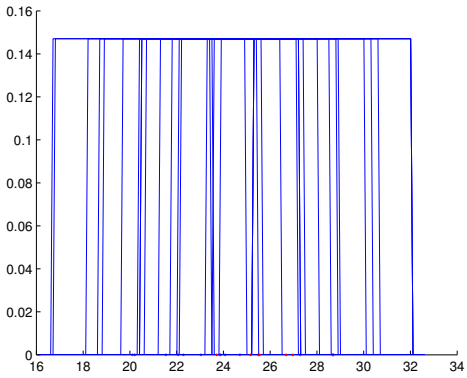
Figura 6: Distribuciones individuales y sumadas. Se han usado 20 datos de una $N(0, 1)$.

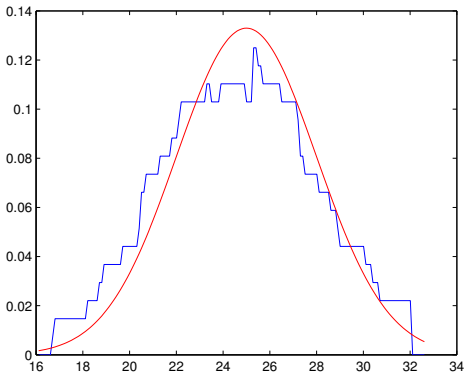
La **desviación típica** de las gausianas, que controla su amplitud, se determinará atendiendo a diferentes criterios, como el número de puntos o la separación entre ellos. Si usamos una σ demasiado grande, se achatará mucho el resultado, **suavizando demasiado** los detalles. En cambio, si usamos una σ demasiado pequeña, habrá muchos picos. Es habitual utilizar un bucle para decidirse por el valor de σ que menor error dé.

```
datos=shuffle([randn(1,1000)*5+30 randn(1,1000)*3+17]);
desvstandard=0.75;
x=5:0.1:45;
h=zeros(1,length(x));
for i=1:length(datos),
  t = normpdf(x,datos(i), desvstandard);
  h = h + t;

  plot(x, t);  hold on;
end

h = h / length(datos);
h2 = (normpdf(x,30,5) + normpdf(x,17,3))/2;

figure, plot(x,h,'r',x,h2,'b');
legend('calculada','exacta')
```

6.6.1. Diferentes distribuciones

El procedimiento de las ventanas de Parzen no solo funciona con distribuciones gausianas, sino que es posible usar otras clases de distribuciones para estimar la función de densidad subyacente. Por ejemplo, es posible utilizar la función de densidad **uniforme** que, centrada en un punto M y usando un ancho T, asigna la misma densidad a los elementos en el rango $[M-\frac{T}{2}, M+\frac{T}{2}]$, y 0 a los elementos fuera de ese rango. Es una especie de función **escalón**. En este caso, el parámetro que habrá que optimizar será el **ancho**.





En Matlab:

```
for i=1:length(x)
  h = h + unifpdf(eje_x, datos(x) - ancho / 2, datos(x) + ancho / 2);
end
```

6.6.2. Cálculo bidimensional

De igual manera que en una dimensión, es posible utilizar el método de las ventanas de Parzen para estimar la función de densidad de probabilidad de una distribución bidimensional. La mecánica es básicamente igual que en una dimensión.

```
function r = Parzen2D(x,y,sigma,xi,yi)
%Parzen2D Estima la func de densidad en los puntos (xi,yi) mediante ventanas de
% Parzen centradas en los datos x,y

% Por cada clase
for i=1:max(y)

    % Aislamos los patrones de esa clase
    xAux = x(:, y == i);

    % Contenedor para las gausianas
    hAux = zeros(1,length(xi(:)));

    % Por cada patron
    for j=1:length(xAux),
        % Agregamos una gausiana centrada en ese patron
        hAux = hAux + mvnpdf([xi(:) yi(:)], xAux(:,j)', [sigma 0;0 sigma]')';
    end

    % Normalizamos y metemos en el contenedor de gausianas
    h(i,:) = hAux' / size(xAux,2);
end

r = h;
end
```

Una posible forma para utilizarlo en un problema de clasificación podría ser la siguiente:

```
[x,y] = shuffle(x,y);

xtrn = x(:, 1:100);
ytrn = y(:, 1:100);

xtst = x(:, 101:end);
ytst = y(:, 101:end);

h = Parzen2D(xtrn, ytrn, 0.7, xtst(1,:), xtst(2,:));

for i=1:3
  % Multiplicamos por las probabilidades a priori
  h(i,:) = h(i,:) * (length(xtrn(ytrn == i)) / length(xtrn));
end

% Nos quedamos con la clase con mayor densidad en cada punto
[~, pos] = max(h);
num_errores = sum(pos ~= ytst);
```