T2.1 Funciones discriminantes

Índice

1 Funciones discriminantes

Notación:

- **Objetos (entradas):** $\boldsymbol{x} \in \mathcal{X}$, típicamente vectores de D características reales, $\mathcal{X} = \mathbb{R}^D$, $D \geq 1$
- **Etiquetas de clase (salidas):** $y \in \mathcal{Y}$ o $c \in \mathcal{C}$, típicamente $\mathcal{Y} = \mathcal{C} = \{1, \ldots, C\}$, $C \geq 2$

Representación clásica de clasificadores: con una **función discriminante** por clase, $g_c(\cdot)$, para medir la (pseudo-)probabilidad de pertenecer a c

$$c(\boldsymbol{x}) = \operatorname*{argmax}_c \; g_c(\boldsymbol{x})$$

Ejemplo: clasificador en 3 clases para $\boldsymbol{x} = (x_1, x_2)^t \in [0, 1]^2$

x_1	x_2	$g_1(\boldsymbol{x})$	$g_2(\boldsymbol{x})$	$g_3(\boldsymbol{x})$	$c(\boldsymbol{x})$
0	0	1.0	0.0	0.0	1
0	1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1
1	0	0.25	0.5	0.25	2
1	1	0.01	0.01	0.98	3

Regla de decisión de Bayes: con $g_c(\boldsymbol{x}) = P(c \mid \boldsymbol{x})$ o $g_c(\boldsymbol{x}) = P(c)\, p(\boldsymbol{x} \mid c)$

2 Funciones discriminantes lineales

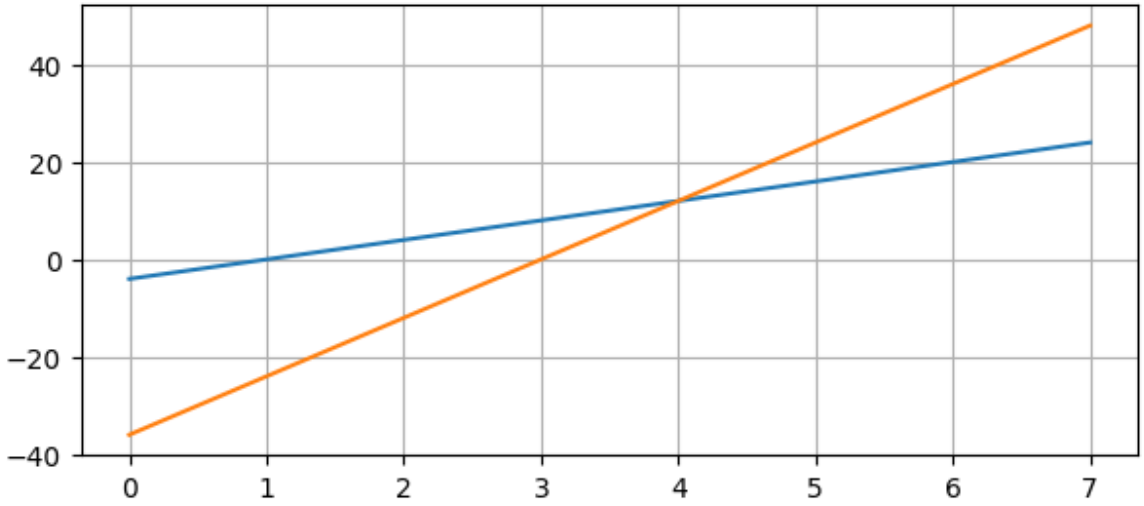
Función discriminante lineal: función lineal con las características

$$g_c(\boldsymbol{x}) = \sum_d w_{cd}\, x_d + w_{c0} = \boldsymbol{w}_c^t\, \boldsymbol{x} + w_{c0}$$

- $\boldsymbol{w}_c = (w_{c1}, w_{c2}, \ldots, w_{cD})^t$ es el **vector de pesos** de la clase c
- w_{c0} es el **peso umbral** de la clase c

Ejemplo: $C = 2$, $x \in \mathbb{R}$, $g_1(x) = 4x - 4$, $g_2(x) = 12x - 36$

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
g1 = lambda x: 4*x-4; g2 = lambda x: 12*x-36; x = np.linspace(0, 7, 2)
plt.figure(figsize=(7, 3)); plt.grid(); plt.plot(x, g1(x), x, g2(x));
```



3 Fronteras de decisión

Frontera de decisión entre clases: lugar geométrico donde las discriminantes de dos (o más) clases empatan

$$g_c(\boldsymbol{x}) = g_{c'}(\boldsymbol{x}) \qquad \text{con } \; c \neq c'$$

Fronteras lineales: las discriminantes lineales originan fronteras lineales de dimensión $D-1$

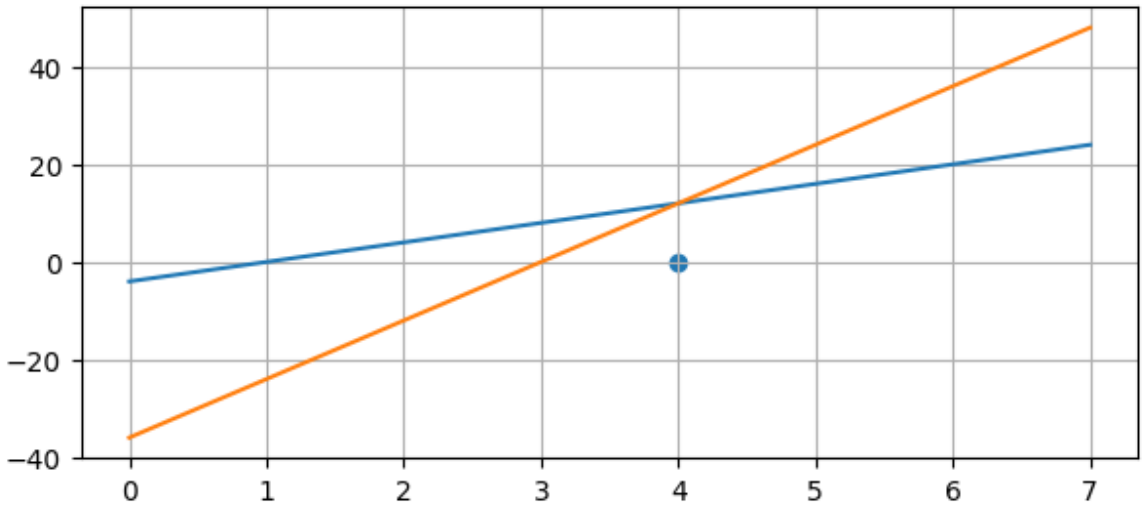
- Un punto, sí $\; \mathcal{X} = \mathbb{R}$
- Una línea (rectas), sí $\; \mathcal{X} = \mathbb{R}^2$
- Una superficie (planos), sí $\; \mathcal{X} = \mathbb{R}^3$

Fronteras en general: **hipersuperfícies** definidas por las ecuaciones

$$g_c(\boldsymbol{x}) - g_{c'}(\boldsymbol{x}) = 0 \qquad \text{con } \; c \neq c'$$

Ejemplo (cont.): $g_1(x) = g_2(x) \rightarrow 4x - 4 = 12x - 36 \rightarrow x = \frac{32}{8} = 4$

In [2]:
```
import numpy as np; import matplotlib.pyplot as plt
g1 = lambda x: 4*x-4; g2 = lambda x: 12*x-36; x = np.linspace(0, 7, 2)
plt.figure(figsize=(7, 3)); plt.grid(); plt.plot(x, g1(x), x, g2(x)); plt.scatter(4, 0);
```



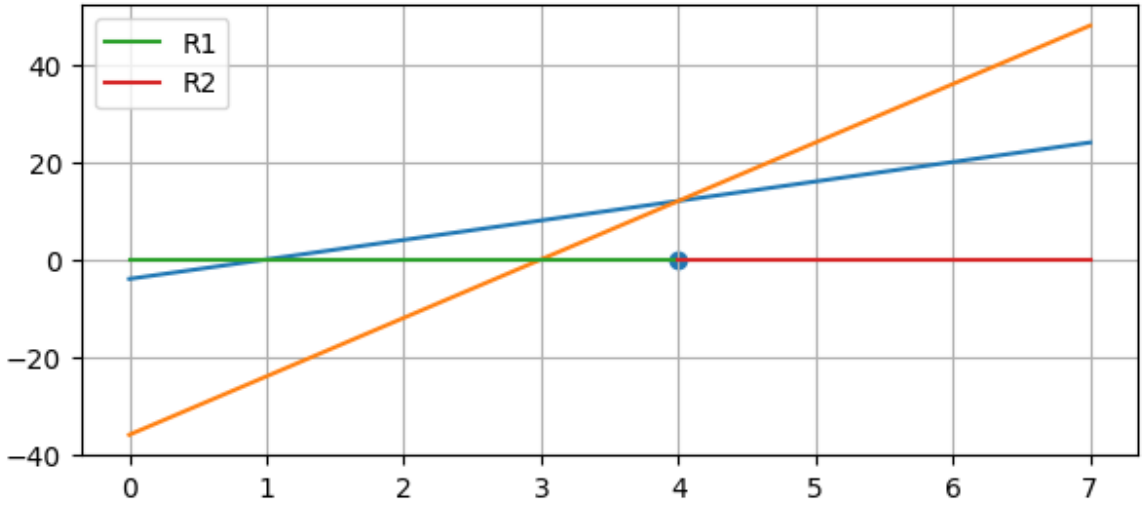
4 Regiones de decisión

Región de decisión de una clase: lugar geométrico donde la discriminante de la clase gana al resto

$$R_c = \{\boldsymbol{x} \in \mathcal{X} : \ g_c(\boldsymbol{x}) > \max_{c' \neq c} g_{c'}(\boldsymbol{x})\}$$

Ejemplo (cont.): $R_1 = \{x : g_1(x) > g_2(x)\} = (-\infty, 4) \quad R_2 = \{x : g_2(x) > g_1(x)\} = (4, \infty)$

In [3]:
```
import numpy as np; import matplotlib.pyplot as plt
g1 = lambda x: 4*x-4; g2 = lambda x: 12*x-36; x = np.linspace(0, 7, 2)
plt.figure(figsize=(7, 3)); plt.grid(); plt.plot(x, g1(x), x, g2(x)); plt.scatter(4, 0)
plt.plot((0, 4), (0, 0), label="R1"); plt.plot((4, 7), (0, 0), label="R2"); plt.legend();
```



5 Clasificadores equivalentes

Propósito: simplificar un clasificador dado, $c(\boldsymbol{x}) = \operatorname{argmax}_c g_c(\boldsymbol{x})$

Clasificador equivalente: $c'(\boldsymbol{x}) = \operatorname{argmax}_c g'_c(\boldsymbol{x})$ es equivalente a $c(\boldsymbol{x})$ si $c(\boldsymbol{x}) = c'(\boldsymbol{x})$ para todo $\boldsymbol{x}$

Construcción: si $f : \mathbb{R} \to \mathbb{R}$ es estrictamente creciente, el siguiente clasificador es equivalente a $c(\boldsymbol{x})$

$$c'(\boldsymbol{x}) = \operatorname{argmax}_c g'_c(\boldsymbol{x}) \quad \text{con} \quad g'_c(\boldsymbol{x}) = f(g_c(\boldsymbol{x})) + \text{const}(\boldsymbol{x})$$

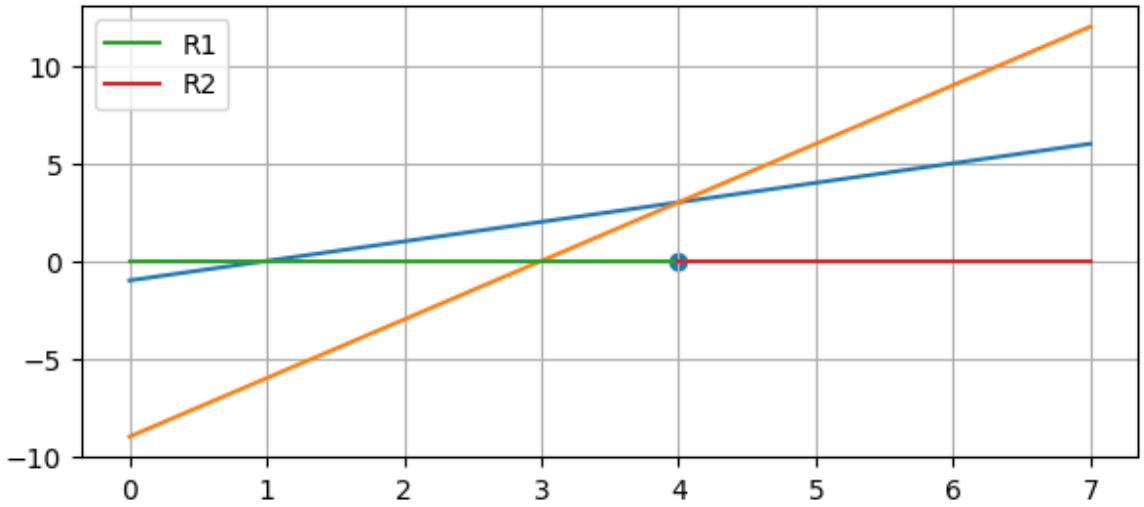
donde $\text{const}(\boldsymbol{x})$ es cualquier función que puede variar (o no) con $\boldsymbol{x}$, pero no con c

Funciones estrictamente crecientes usuales: para construir clasificadores equivalentes simplificados

$$f(z) = az + b \ \text{ con } a > 0 \qquad f(z) = \log z \ \text{ con } z > 0 \qquad f(z) = e^z$$

Ejemplo (cont.): $g_1'(x) = x - 1,\ g_2'(x) = 3x - 9$ con $f(z) = \frac{1}{4}z$

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
g1 = lambda x: x-1; g2 = lambda x: 3*x-9; x = np.linspace(0, 7, 2)
plt.figure(figsize=(7, 3)); plt.grid(); plt.plot(x, g1(x), x, g2(x)); plt.scatter(4, 0)
plt.plot((0, 4), (0, 0), label="R1"); plt.plot((4, 7), (0, 0), label="R2"); plt.legend();
```



Ejercicios T2.1 Funciones discriminantes

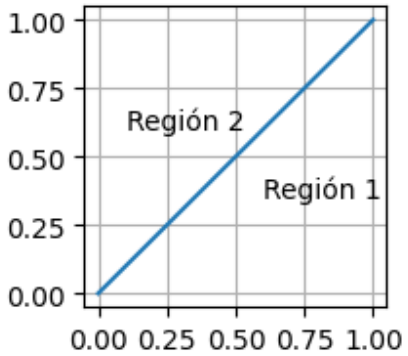
2023_01_26_Cuestión_2: Dado el clasificador en 2 clases definido por sus vectores de pesos $\boldsymbol{w}_1 = (-2, 3, 3)^t,\ \boldsymbol{w}_2 = (0, 2, -2)^t$ en notación homogénea, cúal de los siguientes conjuntos de vectores **no** define un clasificador equivalente al dado?

1. $\boldsymbol{w}_1 = (1, 3, 3)^t,\ \boldsymbol{w}_2 = (3, 2, -2)^t$
2. $\boldsymbol{w}_1 = (-4, 6, 6)^t,\ \boldsymbol{w}_2 = (0, 4, -4)^t$
3. $\boldsymbol{w}_1 = (-1, 6, 6)^t,\ \boldsymbol{w}_2 = (3, 4, -4)^t$
4. $\boldsymbol{w}_1 = (2, -3, -3)^t,\ \boldsymbol{w}_2 = (0, -2, 2)^t$

Solución: la 4.

2023_01_17_Cuestión 3: Sea el clasificador en dos clases definido por la frontera y regiones de decisión de la figura:

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
plt.figure(figsize=(2, 2)); ticks = np.arange(0, 1.1, 0.25); plt.xticks(ticks); plt.yticks(ticks); plt.grid()
plt.plot((0, 1), (0, 1)); plt.text(0.1, 0.6, 'Región 2'); plt.text(0.6, 0.35, 'Región 1');
```



¿Cúal de los siguientes vectores de pesos (en notación homogénea) define un clasificador equivalente al dado?

1. $\boldsymbol{w}_1 = (0, -2, 0)^t$ y $\boldsymbol{w}_2 = (0, 0, -2)^t$.
2. $\boldsymbol{w}_1 = (0, 2, 0)^t$ y $\boldsymbol{w}_2 = (0, 0, 2)^t$.
3. $\boldsymbol{w}_1 = (0, 0, 2)^t$ y $\boldsymbol{w}_2 = (0, 2, 0)^t$.
4. Todos los vectores de pesos anteriores definen clasificadores equivalentes.

Solución: la 2.

2022_02_03_Cuestión: Sea $\boldsymbol{x}$ un objeto a clasificar en una clase de C posibles. Indica qué de los siguientes clasificadores **no** es de error mínimo (o escoge la última opción si los tres son de error mínimo):

1. $c(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\operatorname{argmax}} \ \log p(c) + \log p(\boldsymbol{x}|c) - \log p(\boldsymbol{x})$
2. $c(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\operatorname{argmax}} \ -\log p(\boldsymbol{x}, c)$
3. $c(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\operatorname{argmax}} \ \log p(c) + \log p(\boldsymbol{x}|c)$
4. Los tres clasificadores anteriores son de error mínimo

Solución: la 2.

Problema (cont. de problema T1.2): Considera la clasificación de flores iris en setosa o no-setosa a partir de la longitud de pétalos, x. Se sabe que las distribuciones de x para setosas y no-setosas pueden aproximarse con distribuciones normales de medias y desviaciones estándares:

$$p(x \mid c = \text{set}) \sim \mathcal{N}(\mu_{\text{set}} = 1.46, \sigma_{\text{set}} = 0.17) \qquad \text{y} \qquad p(x \mid c = \text{nos}) \sim \mathcal{N}(\mu_{\text{nos}} = 4.91, \sigma_{\text{nos}} = 0.82)$$

Así mismo, se sabe la probabilidad a priori de setosa es $1/3$ y que la probabilidad a posteriori de que una flor de longitud de pétalos 2 sea setosa es 0.89. Encuentra la frontera de decisión entre setosa y no-setosa, y determina las regiones de decisión de las clases.

Solución:

Frontera de decisión entre las clases setosa y no-setosa:

$$\begin{gathered}
P(c = \text{set} \mid x) = P(c = \text{nos} \mid x) \\
1/3 \cdot \mathcal{N}(x \mid \mu_{\text{set}} = 1.46, \sigma_{\text{set}} = 0.17) = 2/3 \cdot \mathcal{N}(x \mid \mu_{\text{nos}} = 4.91, \sigma_{\text{nos}} = 0.82) \\
\frac{1}{0.17} \exp\left(-\frac{(x - 1.46)^2}{2 \cdot 0.17^2} \right) = \frac{2}{0.82} \exp\left(-\frac{(x - 4.91)^2}{2 \cdot 0.82^2} \right) \\
-17.3\,(x - 1.46)^2 + 0.74\,(x - 4.91)^2 + 0.88 = 0 \\
-17.3x^2 + 50.52x - 36.88 + 0.74x^2 - 7.27x + 17.84 + 0.88 = 0 \\
-16.56x^2 + 43.25x - 18.16 = 0 \\
x = \frac{-43.25 \pm \sqrt{1870.6 - 1202.9}}{-33.12} = \frac{-43.25 \pm 25.84}{-33.12} = \begin{cases} 0.53 \\ 2.09 \end{cases}
\end{gathered}$$

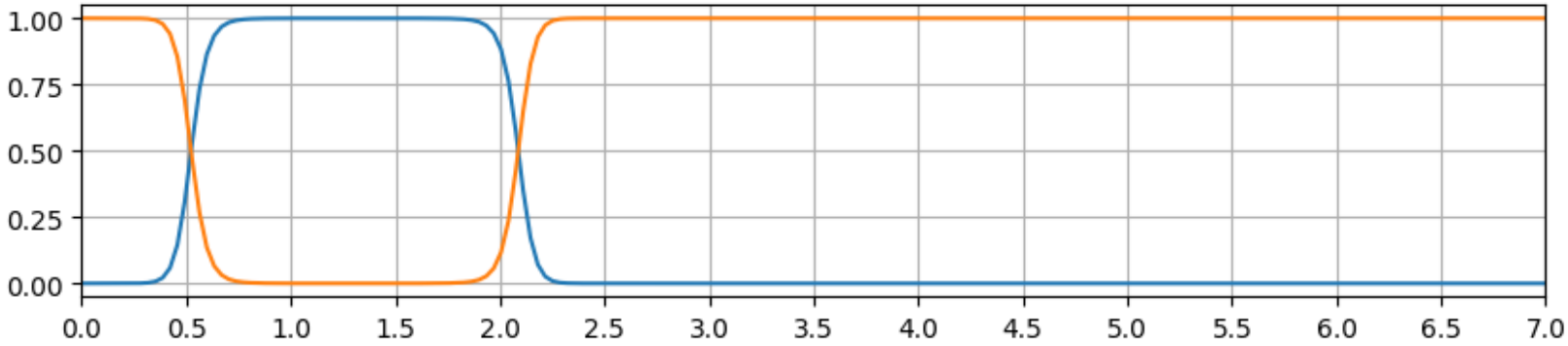
Regiones de decisión:

- Las densidades normales se definen en todo $\mathbb{R}$, pero solo nos interesa $\mathbb{R}^+$
- La frontera encontrada divide $\mathbb{R}^+$ en tres intervalos: $(0, 0.53)$, $(0.53, 2.09)$ y $(2.09, \infty)$
- Dado que $x = 2$ es setosa, $\mathcal{R}_{\text{setosa}} = (0.53, 2.09)$ y $\mathcal{R}_{\text{no-setosa}} = (0, 0.53) \cup (2.09, \infty)$

Dado que la distribución normal es positiva en todo $\mathbb{R}$, la asunción de normalidad de las densidades condicionales implica que asumimos posible cualquier longitud (positiva) de pétalos, tanto de setosas como de no-setosas. Obviamente, es prácticamente imposible encontrarnos setosas o no-setosas de longitud de pétalos en $(0, 0.53)$. Ahora bien, con la asunción hecha, es muy improbable pero no teóricamente imposible. Así, como que la condicional de no-setosa es más plana que la de setosa, los cálculos indican que en $(0, 0.53)$ es más probable que la flor sea no-setosa.

Comprobamos el resultado con gráficas de las probabilidades a posteriori en función de x:

In [3]:
```
import numpy as np; import matplotlib.pyplot as plt; from scipy.stats import norm
p_set = lambda x: norm.pdf(x, 1.46, 0.17); p_nos = lambda x: norm.pdf(x, 4.91, 0.82)
P_set = lambda x: 1/3.0 * p_set(x) / (1/3.0 * p_set(x) + 2/3.0 * p_nos(x))
fig = plt.figure(figsize=(10, 2)); plt.xlim(0, 7); plt.xticks(np.arange(0, 7.1, .5)); plt.grid();
x = np.linspace(0, 7, 200); plt.plot(x, P_set(x), x, 1.0-P_set(x));
```



T2.2 Algoritmo Perceptrón

Índice

1 Algoritmo Perceptrón

Origen: primer algoritmo de aprendizaje automático; propuesto por Frank Rosenblatt en 1958

Entrada: un conjunto de entrenamiento, $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$, con $\boldsymbol{x}_n \in \mathbb{R}^D$ y $y_n \in \{1, \dots, C\}$ para todo n

Salida: los pesos de un clasificador lineal, $c(\boldsymbol{x}) = \operatorname{argmax}_c \; g_c(\boldsymbol{x})$, con $g_c(\boldsymbol{x}) = \boldsymbol{w}_c^t \, \boldsymbol{x} + w_{c0}$ para todo c

Notación homogénea o compacta: $\boldsymbol{x} = (1, x_1, \dots, x_D)^t$ y $\boldsymbol{w}_c = (w_{c0}, w_{c1}, \dots, w_{cD})^t$; así, $g_c(\boldsymbol{x}) = \boldsymbol{w}_c^t \, \boldsymbol{x}$

Objetivo: minimizar el número de errores en entrenamiento

$$\mathcal{L}(\{\boldsymbol{w}_c\}) = \sum\nolimits_n \mathbb{Y}(y_n \neq c(\boldsymbol{x}_n)) = \sum\nolimits_n \mathbb{Y}(\max\nolimits_{c \neq y_n} g_c(\boldsymbol{x}_n) > g_{y_n}(\boldsymbol{x}_n))$$

Objetivo con margen $b \geq 0$: extensión del objetivo básico ($b = 0$) para generalizar mejor

$$\mathcal{L}(\{\boldsymbol{w}_c\}) = \sum\nolimits_n \mathbb{Y}(\max\nolimits_{c \neq y_n} g_c(\boldsymbol{x}_n) + b > g_{y_n}(\boldsymbol{x}_n))$$

Interpretación: la pseudo-probabilidad de pertenecer en la clase correcta tiene que superar la de cualquier clase rival con al menos un margen b

Algoritmo Perceptrón: versión básica con **factor de aprendizaje** $\alpha > 0$ para controlar la velocidad del aprendizaje

> **Entrada:** datos $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$ pesos $\{\boldsymbol{w}_c\}$ factor de aprendizaje $\alpha \in \mathbb{R}^{>0}$ margen $b \in \mathbb{R}^{\geq 0}$
> **Salida:** pesos optimizados $\{\boldsymbol{w}_c\}$
> `repetir`
>
> > `para todo` dato $\boldsymbol{x}_n$
> >
> > > e = `falso`
> > > `para toda` clase $c \neq y_n$
> > >
> > > > `si` $\boldsymbol{w}_c^t \boldsymbol{x}_n + b > \boldsymbol{w}_{y_n}^t \boldsymbol{x}_n$: $\boldsymbol{w}_c = \boldsymbol{w}_c - \alpha \boldsymbol{x}_n$; e = `cierto`
> > >
> > > `si` e: $\boldsymbol{w}_{y_n} = \boldsymbol{w}_{y_n} + \alpha \boldsymbol{x}_n$
>
> `hasta que` no queden muestras mal clasificadas

Implementación: función para problemas de aprendizaje sencillos

In [1]:
```
import numpy as np
def perceptron(X, y, b=0.1, a=1.0, K=200):
    N, D = X.shape; Y = np.unique(y); C = Y.size; W = np.zeros((1+D, C))
    for k in range(1, K+1):
        E = 0
        for n in range(N):
            xn = np.array([1, *X[n, :]])
            cn = np.squeeze(np.where(Y==y[n]))
            gn = W[:,cn].T @ xn; err = False
            for c in np.arange(C):
                if c != cn and W[:,c].T @ xn + b >= gn:
                    W[:, c] = W[:, c] - a*xn; err = True
            if err:
                W[:, cn] = W[:, cn] + a*xn; E = E + 1
        if E == 0:
            break;
    return W, E, k
```

2 Ejemplo

Entrada: $C = D = 2 \quad \boldsymbol{x}_1 = (1,0,0)^t \quad y_1 = 1 \quad \boldsymbol{x}_2 = (1,1,1)^t \quad y_2 = 2 \quad \alpha = 1 \quad b = 0.1$

Traza:

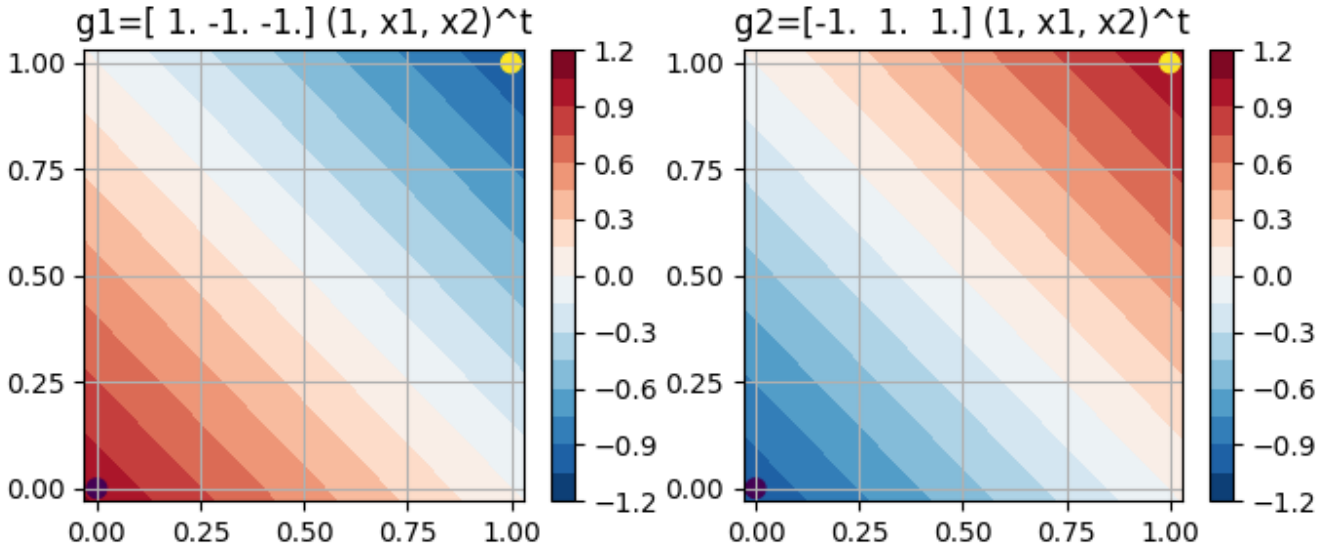
Iteración	n	$\boldsymbol{w}_{y_n}^t \boldsymbol{x}_n$	$c, c \neq y_n$	$\boldsymbol{w}_c^t \boldsymbol{x}_n + b$	$\boldsymbol{w}_1^t$	$\boldsymbol{w}_2^t$
					$(0,0,0)$	$(0,0,0)$
1	1	$(0,0,0)(1,0,0)^t = 0$	2	$(0,0,0)(1,0,0)^t + 0.1 = 0.1$		$(-1,0,0)$
1	1				$(1,0,0)$	
1	2	$(-1,0,0)(1,1,1)^t = -1$	1	$(1,0,0)(1,1,1)^t + 0.1 = 1.1$	$(0,-1,-1)$	
1	2					$(0,1,1)$
2	1	$(0,-1,-1)(1,0,0)^t = 0$	2	$(0,1,1)(1,0,0)^t + 0.1 = 0.1$		$(-1,1,1)$
2	1				$(1,-1,-1)$	
2	2	$(-1,1,1)(1,1,1)^t = 1$	1	$(1,-1,-1)(1,1,1)^t + 0.1 = -0.9$		
3	1	$(1,-1,-1)(1,0,0)^t = 1$	2	$(-1,1,1)(1,0,0)^t + 0.1 = -0.9$		
3	2	$(-1,1,1)(1,1,1)^t = 1$	1	$(1,-1,-1)(1,1,1)^t + 0.1 = -0.9$		

In [2]:
```
X = np.array([[0, 0], [1, 1]]); y = np.array([0, 1], dtype=int);
W, E, k = perceptron(X, y)
print("w1 =", W[:,0].T, "  w2 =", W[:,1].T, "  E =", E, "  k =", k)
```

```
w1 = [ 1. -1. -1.]   w2 = [-1.  1.  1.]   E = 0   k = 3
```

Frontera: $\boldsymbol{w}_1^t(1, x_1, x_2)^t = \boldsymbol{w}_2^t(1, x_1, x_2)^t \;\rightarrow\; x_2 = -x_1 + 1$

In [3]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[0, 0], [1, 1]]).astype(float); y = np.array([1, 2]).astype(int)
x1, x2 = np.meshgrid(np.linspace(-.03, 1.03, 50), np.linspace(-.03, 1.03, 50))
XX = np.c_[np.ravel(x1), np.ravel(x2)]
Wt = np.array([[1, -1, -1], [-1, 1, 1]]).astype(float)
gg = lambda x: (Wt[0, 0] + Wt[0, 1:] @ x, Wt[1, 0] + Wt[1, 1:] @ x)
GG = np.apply_along_axis(gg, 1, XX)
_, axs = plt.subplots(1, 2, figsize=(8, 3))
for i, ax in enumerate(axs.flat):
    ax.set_xticks(np.linspace(0., 1, 5)); ax.set_yticks(np.linspace(0., 1, 5))
    ax.grid(); ax.set_title(f'g{i+1}={W[:,i]} (1, x1, x2)^t')
    cp = ax.contourf(x1, x2, GG[:, i].reshape(x1.shape), 15, cmap='RdBu_r')
    plt.colorbar(cp, ax=ax); ax.scatter(*X.T, c=y, s=50);
```



3 Convergencia y calidad de la solución

Convergencia: Perceptrón converge si los datos son **linealmente separables**

Efecto del factor de aprendizaje $\alpha > 0$: converge con independencia del valor elegido, aunque lentamente si α es muy pequeño

Efecto del margen $b \geq 0$: converge con fronteras centradas si elegimos un valor próximo al máximo margen que permite discriminar linealmente las muestras con margen; si nos pasamos, Perceptró no converge

- Hay que hacer experimentos con diferentes valores del margen para encontrar un valor que generalice óptimamente

Ejercicios T2.2 Algoritmo Perceptrón

2023_01_26_Problema: La tabla siguiente incluye un conjunto de 4 muestras bidimensionales de aprendizaje de 3 clases, $c = 1, 2, 3$.

n	x_{n1}	x_{n2}	c_n
1	3	5	3
2	5	1	1
3	2	2	1
4	1	2	2

Se pide:

1. (1.5 puntos) Realiza una traza de ejecución de una iteración del algoritmo Perceptrón, con factor de aprendizaje $\alpha = 1$ margen $b = 0.1$ y los siguientes pesos iniciales de cada clase por columnas:

d	w_{d1}	w_{d2}	w_{d3}
0	-1	0	-3
1	4	-6	-8
2	-10	-2	2

2. (0.5 puntos) Clasifica la muestra de test $\boldsymbol{x} = (5, 5)^t$ mediante un clasificador lineal con los vectorss de pesos obtenidos en el apartado anterior.

Solución:

1. Una iteración de Perceptrón con 1 muestra mal clasificada y pesos resultantes:

d	w_{d1}	w_{d2}	w_{d3}
0	-1	1	-4
1	4	-5	-9
2	-10	0	0

2. Clasificación de la muestra de test: $g_1(\boldsymbol{x}) = -31 \quad g_2(\boldsymbol{x}) = -24 \quad g_3(\boldsymbol{x}) = -49 \quad \Rightarrow \quad c(\boldsymbol{x}) = 2$

2023_01_17_Cuestión 4: Supón que estamos aplicando el algoritmo Perceptrón, con factor de aprendizaje $\alpha = 1$ y margen $b = 0.1$, a un conjunto de 4 muestras bidimensionales de aprendizaje para un problema de 4 clases, $c = 1, 2, 3, 4$. En un momento dado de la ejecución del algoritmo se han obtenido los vectores de pesos $\boldsymbol{w}_1 = (-2, -2, -6)^t$, $\boldsymbol{w}_2 = (-2, -2, -6)^t$, $\boldsymbol{w}_3 = (-2, -4, -4)^t$, $\boldsymbol{w}_4 = (-2, -4, -4)^t$. Suponiendo que a continuación se va a procesar la muestra $(\boldsymbol{x}, c) = ((4, 5)^t, 2)$, cuántos vectores de pesos se modificarán?

1. 0
2. 2
3. 3
4. 4

Solución: la 4.

2022_01_27_Problema: En la tabla de la izquierda se proporciona un conjunto de 3 muestras bidimensionales de aprendizaje de 3 clases, mientras que en la tabla de la derecha se proporciona un conjunto de pesos iniciales por cada clase.

n	x_{n1}	x_{n2}	c_n
1	-2	-2	1
2	0	0	2
3	2	2	3

	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$
w_{c0}	0	-1	-1
w_{c1}	-2	0	4
w_{c2}	-2	0	4

Se pide:

1. (1.5 puntos) Realiza una traza de ejecución de una iteración del algoritmo Perceptrón, con factor de aprendizaje $\alpha = 1$, margen $\gamma = 0.1$ y utilizando los pesos iniciales proporcionados.
2. (0.5 puntos) Representa las regiones de decisión del clasificador resultante, así como las fronteras de decisión necesarias para su representación.

Solución: Una iteración de Perceptrón con 1 muestra mal clasificada obtiene los siguientes pesos finales:

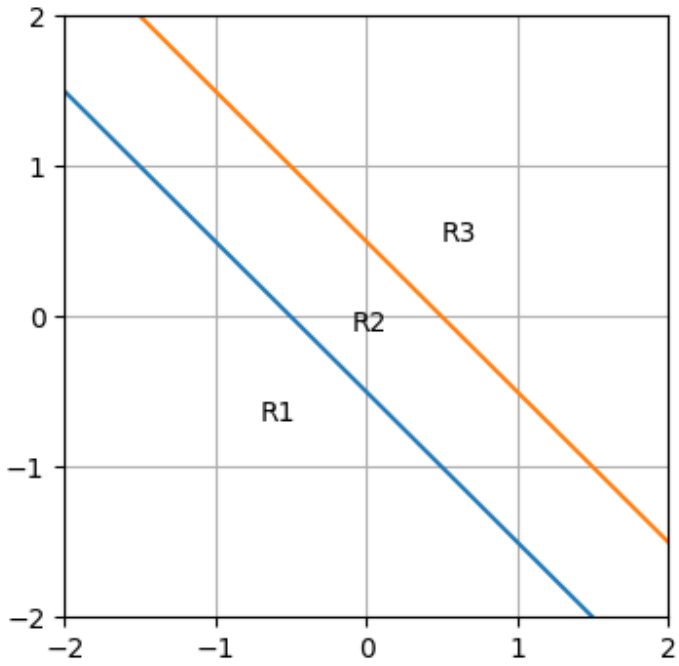
	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$
w_{c0}	-1	0	-2
w_{c1}	-2	0	4
w_{c2}	-2	0	4

Fronteras:

$$\boldsymbol{w}_1^t\boldsymbol{x} = \boldsymbol{w}_2^t\boldsymbol{x} \;\Rightarrow\; -1 - 2x_1 - 2x_2 = 0 \;\Rightarrow\; x_2 = -x_1 - 0.5$$
$$\boldsymbol{w}_1^t\boldsymbol{x} = \boldsymbol{w}_3^t\boldsymbol{x} \;\Rightarrow\; -1 - 2x_1 - 2x_2 = -2 + 4x_1 + 4x_2 \;\Rightarrow\; x_2 = -x_1 + \frac{1}{6}$$
$$\boldsymbol{w}_2^t\boldsymbol{x} = \boldsymbol{w}_3^t\boldsymbol{x} \;\Rightarrow\; 0 = -2 + 4x_1 + 4x_2 \;\Rightarrow\; x_2 = -x_1 + 0.5$$

La representación gráfica de las tres regiones de decisión con las dos fronteras de decisión involucradas es la siguiente:

In [24]:
```
import numpy as np; import matplotlib.pyplot as plt
fig = plt.figure(figsize=(4, 4)); plt.xlim([-2, 2]); plt.ylim([-2, 2])
ticks = np.arange(-2, 2.1); plt.xticks(ticks); plt.yticks(ticks); plt.grid()
plt.plot((-2, 2), (1.5, -2.5)); # frontera R1 - R2
plt.plot((-2, 2), (2.5, -1.5)); # frontera R2 - R3
plt.text(-.7, -.7, 'R1'); plt.text(-.1, -.1, 'R2'); plt.text(.5, .5, 'R3');
```



2022_01_13_Cuestión 3: Supón que estamos aplicando el algoritmo Perceptrón, con factor de aprendizaje $\alpha = 1$ y margen $b = 0.1$, a un conjunto de 3 muestras bidimensionales de aprendizaje para un problema de 2 clases. Se sabe que, después de procesar las primeras 2 muestras, se han obtenido los vectores de pesos $\boldsymbol{w}_1 = (0, 0, 1)^t$, $\boldsymbol{w}_2 = (0, 0, -1)^t$. Así mismo, se sabe que, después de procesar la última muestra, $(\boldsymbol{x}_3, c_3)$, se obtienen los vectores de pesos $\boldsymbol{w}_1 = (-1, -5, -2)^t$, y $\boldsymbol{w}_2 = (1, 5, 2)^t$. ¿Cuál de las siguientes muestras es esa última muestra?

1. $((2, 4)^t, 2)$
2. $((5, 4)^t, 2)$
3. $((5, 3)^t, 2)$
4. $((2, 5)^t, 2)$

Solución: la 3.

T2.3 Estimación del error

Índice

1 Error teórico de un clasificador

Propósito: determinar la probabilidad de error de un clasificador dado, $c(\boldsymbol{x}) = \operatorname{argmax}_c g_c(\boldsymbol{x})$

Error a posteriori: probabilidad de error para un $\boldsymbol{x}$ dado; uno menos la probabilidad de acertar

$$\varepsilon(c(\boldsymbol{x})) = 1 - P(c(\boldsymbol{x}) \mid \boldsymbol{x})$$

Error (a priori) con $\mathcal{X}$ discreto: probabilidad de error esperada de acuerdo con la **probabilidad incondicional** de $\boldsymbol{x}$

$$\varepsilon = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c(\boldsymbol{x}))] = \sum_{\boldsymbol{x}} P(\boldsymbol{x})\, \varepsilon(c(\boldsymbol{x}))$$

Ejemplo: $\mathcal{X} = \{0,1\}^2$, $C = 2$

Conocimiento teórico

x_1	x_2	$P(\boldsymbol{x})$	$P(1\|\boldsymbol{x})$	$P(2\|\boldsymbol{x})$
0	0	1/2	1	0
0	1	1/4	3/4	1/4
1	0	1/4	1/4	3/4
1	1	0	0	1

Error teórico de un clasificador dado

x_1	x_2	$c(\boldsymbol{x})$	$\varepsilon(c(\boldsymbol{x}))$	$P(\boldsymbol{x})\varepsilon(c(\boldsymbol{x}))$
0	0	1	0	0
0	1	1	1/4	1/16
1	0	1	3/4	3/16
1	1	2	0	0

$$\varepsilon = 1/16 + 3/16 = 1/4$$

Error (a priori) con $\mathcal{X}$ continuo: probabilidad de error esperada de acuerdo con la **densidad de probabilidad incondicional** de $\boldsymbol{x}$

$$\varepsilon = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c(\boldsymbol{x}))] = \int p(\boldsymbol{x})\, \varepsilon(c(\boldsymbol{x}))\, d\boldsymbol{x}$$

Ejemplo: $\mathcal{X} = \mathbb{R} \quad C = 2 \quad c(x) = 1$

Conocimiento teórico: $p(x) = \begin{cases} 1/2 & x \in [-3/4, -1/4] \\ 1 & x \in [-1/4, 1/4] \\ 1/2 & x \in [1/4, 3/4] \end{cases} \qquad P(c=1|x) = \begin{cases} 1 & x \in [-3/4, -1/4] \\ 1/2 & x \in [-1/4, 1/4] \\ 0 & x \in [1/4, 3/4] \end{cases}$

Error teórico del clasificador dado:

$$\varepsilon(c(x)) = \begin{cases} 0 & x \in [-3/4, -1/4] \\ 1/2 & x \in [-1/4, 1/4] \\ 1 & x \in [1/4, 3/4] \end{cases} \rightarrow \varepsilon = \int p(x)\, \varepsilon(c(x))\, dx = \int_{-1/4}^{1/4} 1 \cdot \frac{1}{2}\, dx + \int_{1/4}^{3/4} \frac{1}{2} \cdot 1\, dx = \frac{1}{2}$$

2 Error del clasificador de Bayes

Clasificador de Bayes o regla de decisión MAP (máximo a posteriori): $c^*(\boldsymbol{x}) = \operatorname{argmax}_c \; P(c \mid \boldsymbol{x})$

Error de Bayes a posteriori: $\varepsilon(c^*(\boldsymbol{x})) = 1 - P(c^*(\boldsymbol{x}) \mid \boldsymbol{x}) = 1 - \max_c P(c \mid \boldsymbol{x})$

Error de Bayes (a priori) con $\mathcal{X}$ discreto: $\varepsilon^* = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c^*(\boldsymbol{x}))] = \sum_{\boldsymbol{x}} P(\boldsymbol{x})\, \varepsilon(c^*(\boldsymbol{x}))$

Ejemplo (cont.): $\mathcal{X} = \{0,1\}^2$, $C = 2$

Conocimiento teórico

x_1	x_2	$P(\boldsymbol{x})$	$P(1\|\boldsymbol{x})$	$P(2\|\boldsymbol{x})$
0	0	1/2	1	0
0	1	1/4	3/4	1/4
1	0	1/4	1/4	3/4
1	1	0	0	1

Error de Bayes

x_1	x_2	$c^*(\boldsymbol{x})$	$\varepsilon(c^*(\boldsymbol{x}))$	$P(\boldsymbol{x})\varepsilon(c^*(\boldsymbol{x}))$
0	0	1	0	0
0	1	1	1/4	1/16
1	0	2	1/4	1/16
1	1	2	0	0

$$\varepsilon^* = 1/16 + 1/16 = 1/8$$

Error de Bayes (a priori) con $\mathcal{X}$ continuo: $\varepsilon^* = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c^*(\boldsymbol{x}))] = \int p(\boldsymbol{x})\, \varepsilon(c^*(\boldsymbol{x}))\, d\boldsymbol{x}$

Ejemplo (cont.): $\mathcal{X} = \mathbb{R} \quad C = 2 \quad c^*(x) = 1 + \mathbb{I}(1/4 \leq x \leq 3/4)$

Conocimiento teórico: $p(x) = \begin{cases} 1/2 & x \in [-3/4, -1/4] \\ 1 & x \in [-1/4, 1/4] \\ 1/2 & x \in [1/4, 3/4] \end{cases} \qquad P(c = 1 \mid x) = \begin{cases} 1 & x \in [-3/4, -1/4] \\ 1/2 & x \in [-1/4, 1/4] \\ 0 & x \in [1/4, 3/4] \end{cases}$

Error de Bayes: $\varepsilon(c^*(x)) = \begin{cases} 0 & x \in [-3/4, -1/4] \\ 1/2 & x \in [-1/4, 1/4] \\ 0 & x \in [1/4, 3/4] \end{cases} \rightarrow \varepsilon^* = \int p(x)\, \varepsilon(c^*(x))\, dx = \int_{-1/4}^{1/4} 1 \cdot \frac{1}{2}\, dx = \frac{1}{4}$

3 Estimación del error de un clasificador

Imposibilidad de calcular el error teórico exacto: puesto que no tenemos el conocimiento teórico exacto

Estimación del error por resubstitución: consiste en emplear la tasa de error con los datos de entrenamiento

Resubstitución es optimista: un clasificador que se limite a clasificar bien los datos de entrenamiento (por ejemplo mediante memorización) obtendrá un estimación del error (casi) nula, pero funcionará mal con datos futuros, no vistos en entrenamiento

Estimación del error por holdout: consiste a "dejar fuera" del entrenamiento una parte de los datos disponibles, la cual denominamos **conjunto de test,** y emplear la tasa de error con los datos de test

Holdout es (ligeramente) pesimista: puesto que estima el error de un clasificador entrenado con menos datos de los disponibles y, en general, la tasa de error empeora al reducir el número de datos de entrenamiento

Aproximación normal a la distribución del estimador holdout: si el número de datos de test, M, es grande, el estimador holdout puede considerarse una variable aleatoria normal de media el error teórico que queremos estimar y varianza inversamente proporcional a M

$$\hat{\varepsilon} \sim \mathcal{N}\left(\varepsilon, \frac{\varepsilon(1-\varepsilon)}{M}\right)$$

Intervalo de confianza al 95%: la aproximación normal permite acompañar la estimación del error con un intervalo de confianza al 95%

$$I_{95\%} = [\hat{\varepsilon} - \hat{r}, \hat{\varepsilon} + \hat{r}] \quad \text{con radio} \quad \hat{r} = 1.96\sqrt{\frac{\hat{\varepsilon}(1-\hat{\varepsilon})}{M}}$$

Ejemplo: si obtenemos 100 errores al clasificar $M = 2000$ muestras de test

$$\hat{\varepsilon} = \frac{100}{2000} = 0.05 = 5\%$$
$$\hat{r} = 1.96\sqrt{\frac{0.05 \cdot 0.95}{2000}} = 0.01$$
$$I_{95\%} = [0.04, 0.06] = [4\%, 6\%]$$

Ejercicios T2.3 Estimación del error

2023_01_17_Cuestión 2: Sea un problema de clasificación en cuatro clases para datos del tipo $\boldsymbol{x} = (x_1, x_2)^t \in \{0,1\}^2$, con las distribuciones de probabilidad de la tabla:

x_1	x_2	$P(c=1\|\boldsymbol{x})$	$P(c=2\|\boldsymbol{x})$	$P(c=3\|\boldsymbol{x})$	$P(c=4\|\boldsymbol{x})$	$P(\boldsymbol{x})$
0	0	0.1	0.3	0.1	0.5	0
0	1	0.2	0.5	0.3	0	0.1
1	0	0.2	0.4	0.1	0.3	0.3
1	1	0.1	0.3	0.3	0.3	0.6

Indica en qué intervalo se encuentra el error de Bayes, ε^*:

1. $\varepsilon^* < 0.40$.
2. $0.40 \leq \varepsilon^* < 0.45$.
3. $0.45 \leq \varepsilon^* < 0.50$.
4. $0.50 \leq \varepsilon^*$.

Solución: $\varepsilon^* = 0.65$

2022_01_27_Cuestión 3: Sea un problema de clasificación en tres clases para datos del tipo $\boldsymbol{x} = (x_1, x_2)^t \in \{0,1\}^2$, con las distribuciones de probabilidad de la tabla:

x_1	x_2	$P(c=1 \mid \boldsymbol{x})$	$P(c=2 \mid \boldsymbol{x})$	$P(c=3 \mid \boldsymbol{x})$	$P(\boldsymbol{x})$	$c(\boldsymbol{x})$
0	0	0.2	0.1	0.7	0.2	2
0	1	0.4	0.3	0.3	0	1
1	0	0.3	0.4	0.3	0.4	3
1	1	0.4	0.4	0.2	0.4	1

Indica en qué intervalo se encuentra el error del clasificador $c(\boldsymbol{x})$ dado en la tabla, ε:

1. $\varepsilon < 0.25$
2. $0.25 \leq \varepsilon < 0.50$
3. $0.50 \leq \varepsilon < 0.75$
4. $0.75 \leq \varepsilon$

Solución: la 3 ya que $\varepsilon = 0.70$

2022_01_13_Cuestión 2: La probabilidad de error de un clasificador se estima que es del 20%. Determina cuál es el número mínimo de muestras de test necesarias, M, para conseguir que el intervalo de confianza al 95% del dicho error no supere el $\pm 1\%$; esto es, $Y = [19\%, 21\%]$:

1. $M < 2000$
2. $2000 \leq M < 3500$
3. $3500 \leq M < 5000$
4. $M \geq 5000$

Solución: la 4 ya que $M = 6147$

T2.4 Regresión logística

Índice

1 Codificación one-hot y distribución categórica

Variable categórica: variable aleatoria que toma un valor de un conjunto finito de categorías (no ordenadas)

Ejemplos de variables categóricas: color RGB, **etiqueta de clase,** palabra de un vocabulario, etc.

Codificación one-hot: de una variable categórica y que toma un valor entre C posibles, $\{1, \dots, C\}$

$$\text{one-hot}(y) = \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_C \end{pmatrix} = \begin{pmatrix} \mathbb{I}(y=1) \\ \vdots \\ \mathbb{I}(y=C) \end{pmatrix} \in \{0,1\}^C \quad \text{amb} \quad \sum_c y_c = 1$$

Distribución categórica: distribución de probabilidades entre las C posibles categorías de una variable categórica, las probabilidades de las cuales vienen dadas por un vector parámetros $\boldsymbol{\theta} \in [0,1]^C$ tal que $\sum_c \theta_c = 1$

$$\text{Cat}(y \mid \boldsymbol{\theta}) = \prod_{c=1}^{C} \theta_c^{\mathbb{I}(y=c)} \qquad \text{o, en notació one-hot,} \qquad \text{Cat}(\boldsymbol{y} \mid \boldsymbol{\theta}) = \prod_{c=1}^{C} \theta_c^{y_c}$$

Convención: $0^0 = 1$ y $0 \log 0 = 0$; por ejemplo, con $\boldsymbol{\theta} = (0.5, 0.5, 0)^t$, $\text{Cat}(\boldsymbol{y} = (1,0,0)^t \mid \boldsymbol{\theta}) = 0.5^1 0.5^0 0^0 = 0.5$

2 Modelo probabilístico de clasificación con softmax

Normalización probabilística de clasificadores: todo clasificador definido con funciones discriminantes generales puede representarse mediante un clasificador equivalente con funciones discriminantes normalizadas probabilísticamente

$$\begin{aligned} c(\boldsymbol{x}) &= \operatorname*{argmax}_c \ a_c && \text{donde } a_c \text{ es la discriminante de la clase } c \text{ evaluada con } \boldsymbol{x} \\ &= \operatorname*{argmax}_c \ e^{a_c} && \text{con } h(z) = e^z \in \mathbb{R}^{\geq 0} \text{ estrictamente creciente} \\ &= \operatorname*{argmax}_c \ \frac{e^{a_c}}{\sum_{c'} e^{a_{c'}}} && \text{con } h(z) = kz,\ k \text{ constante positiva (invariable con } c) \end{aligned}$$

La función softmax: transforma un vector de **logits** (log-probabilidades no normalizadas) $\boldsymbol{a} \in \mathbb{R}^C$ en uno de probabilidades $[0,1]^C$

$$\mathcal{S}(\boldsymbol{a}) = \left[\frac{e^{a_1}}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}}, \ldots, \frac{e^{a_C}}{\sum_{\tilde{c}} e^{a_{\tilde{c}}}}\right] \qquad \text{cumpliéndose} \qquad 0 \leq \mathcal{S}(\boldsymbol{a})_c \leq 1 \quad \text{y} \quad \sum_c \mathcal{S}(\boldsymbol{a})_c = 1$$

Modelo probabilístico de clasificación con softmax: en vez de predecir una única clase más probable, predecimos las probabilidades de todas las clases a partir de una función predictora de logits, $f : \mathcal{X} \to \mathbb{R}^C$, gobernada por un vector de parámetros $\boldsymbol{\theta}$

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathrm{Cat}(\boldsymbol{y} \mid \mathcal{S}(f(\boldsymbol{x}; \boldsymbol{\theta}))) = \prod_c (\mathcal{S}(f(\boldsymbol{x}; \boldsymbol{\theta}))_c)^{y_c}$$

Conveniencia del modelo en inferencia: la predicción de las probabilidades de todas las clases permite aplicar reglas más generales que la MAP, por ejemplo en caso de errores con costes diferentes; además, si queremos aplicar la regla MAP, no hace falta softmax-normalizar logits

Conveniencia del modelo en aprendizaje: permite plantear el aprendizaje probabilísticamente, con criterios estándares como por ejemplo máxima verosimilitud; además, gracias a la softmax, $f(\boldsymbol{x};\boldsymbol{\theta})$ puede elegirse libremente puesto que no está sujeta a las restricciones de probabilidad

3 Regresión logística

Regresión logística: modelo con softmax y **logits lineales** con la entrada (en notación homogénea)

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \mathbf{W}) = \mathrm{Cat}(\boldsymbol{y} \mid \boldsymbol{\mu}) \quad \text{con} \quad \boldsymbol{\mu} = \mathcal{S}(\boldsymbol{a}), \quad \boldsymbol{a} = f(\boldsymbol{x}; \mathbf{W}) = \mathbf{W}^t \boldsymbol{x}, \quad \mathbf{W} \in \mathbb{R}^{(D+1)\times C} \quad \text{y} \quad \boldsymbol{x} \in \mathbb{R}^D$$

Diferencia con los clasificadores basados en discriminantes lineales: ninguno, salvo que ahora predecimos las probabilidades de todas las clases

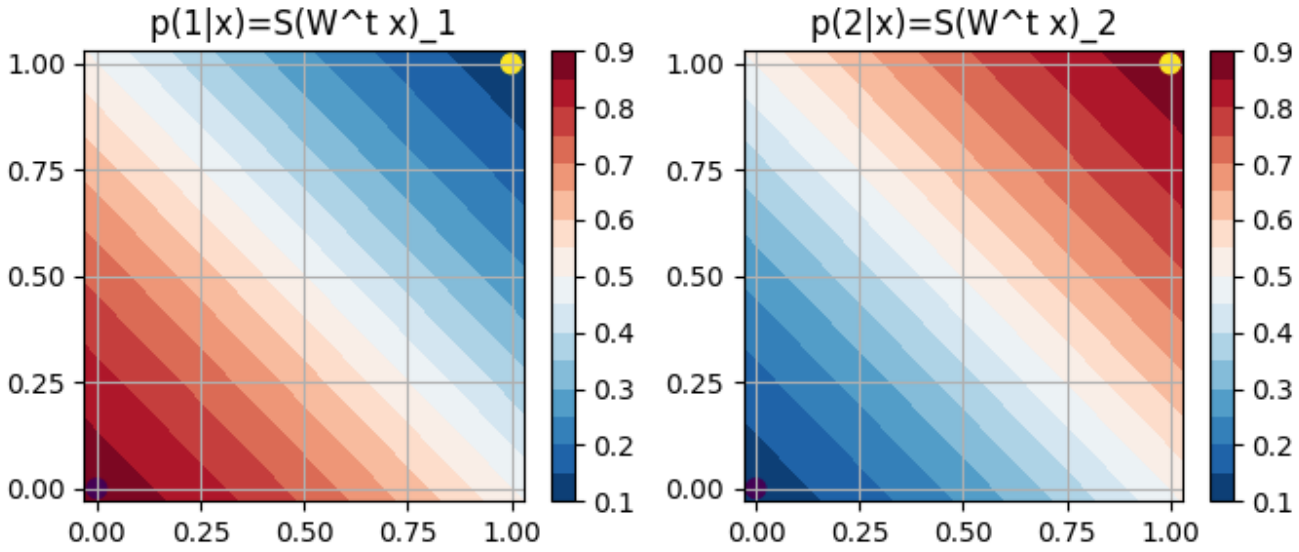
Ejemplo (cont. de Perceptrón):

$$C = D = 2, \quad a_1 = g_1(x_1, x_2) = -x_1 - x_2 + 1, \quad a_2 = g_2(x_1, x_2) = x_1 + x_2 - 1$$

$$\boldsymbol{a} = f(\boldsymbol{x}; \mathbf{W}) = \mathbf{W}^t \boldsymbol{x} \quad \text{con} \quad \mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix} \quad \text{y} \quad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$

$\boldsymbol{x}^t$	$\boldsymbol{a}^t$	$\mu_1 = \mathcal{S}(\boldsymbol{a})_1$	$\mu_2 = \mathcal{S}(\boldsymbol{a})_2$
$(1, 0, 0)$	$(1, -1)$	$\frac{e^1}{e^1+e^{-1}} = 0.8808$	$\frac{e^{-1}}{e^1+e^{-1}} = 0.1192$
$(1, 1, 1)$	$(-1, 1)$	$\frac{e^{-1}}{e^{-1}+e^1} = 0.1192$	$\frac{e^1}{e^{-1}+e^1} = 0.8808$
$(1, 0.5, 0.5)$	$(0, 0)$	$\frac{e^0}{e^0+e^0} = 0.5000$	$\frac{e^0}{e^0+e^0} = 0.5000$

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[0, 0], [1, 1]]).astype(float); y = np.array([1, 2]).astype(int)
x1, x2 = np.meshgrid(np.linspace(-.03, 1.03, 50), np.linspace(-.03, 1.03, 50))
XX = np.c_[np.ravel(x1), np.ravel(x2)]
Wt = np.array([[1, -1, -1], [-1, 1, 1]]).astype(float)
P = lambda x: (np.exp(Wt[0, 0] + Wt[0, 1:] @ x), np.exp(Wt[1, 0] + Wt[1, 1:] @ x))
PP = np.apply_along_axis(P, 1, XX); PP = PP/PP.sum(axis=1, keepdims=True)
_, axs = plt.subplots(1, 2, figsize=(8, 3))
for i, ax in enumerate(axs.flat):
    ax.set_xticks(np.linspace(0., 1, 5)); ax.set_yticks(np.linspace(0., 1, 5));
    ax.grid(); ax.set_title(f'p({i+1}|x)=S(W^t x)_{i+1}')
    cp = ax.contourf(x1, x2, PP[:, i].reshape(x1.shape), 15, cmap='RdBu_r')
    plt.colorbar(cp, ax=ax); ax.scatter(*X.T, c=y, s=50);
```



4 Aprendizaje por máxima verosimilitud

Propósito: establecer un criterio para aprender $\mathbf{W}$ a partir de un conjunto de datos de entrenamiento, $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$

Aprendizaje por máxima verosimilitud

Log-verosimilitud (condicional): log-probabilidad de $\mathcal{D}$ interpretada como función de $\mathbf{W}$

$$\begin{aligned}\mathrm{LL}(\mathbf{W}) &= \log p(\mathcal{D} \mid \mathbf{W}) = \log \prod_{n=1}^N p(\boldsymbol{y}_n \mid \boldsymbol{x}_n, \mathbf{W}) \\ &= \sum_{n=1}^N \log \mathrm{Cat}(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n) \qquad \text{con} \quad \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n) \quad \text{y} \quad \boldsymbol{a}_n = \mathbf{W}^t \boldsymbol{x}_n \\ &= \sum_{n=1}^N \log \prod_{c=1}^C \mu_{nc}^{y_{nc}} = \sum_{n=1}^N \sum_{c=1}^C y_{nc} \log \mu_{nc}\end{aligned}$$

Ejemplo (cont.): log-verosimilitud de $\mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$ amb $\mathcal{D} = \{((1,0,0)^t, (1,0)^t), ((1,1,1)^t, (0,1)^t)\}$

$$\begin{aligned}\mathrm{LL}(\mathbf{W}) &= y_{11} \log \mu_{11} + y_{12} \log \mu_{12} + y_{21} \log \mu_{21} + y_{22} \log \mu_{22} \\ &= \log \mu_{11} + \log \mu_{22} \\ &= \log 0.8808 + \log 0.8808 = -0.1269 - 0.1269 = -0.2538\end{aligned}$$

Aprendizaje por máxima verosimilitud: elegimos un $\mathbf{W}$ que otorga máxima probabilidad a $\mathcal{D}$

$$\mathbf{W}^* = \operatorname*{argmax}_{\mathbf{W}} \mathrm{LL}(\mathbf{W})$$

Planteamiento como un problema de minimización

Neg-log-verosimilitud: log-verosimilitud con el signo cambiado y normalizada por el número de datos

$$\mathrm{NLL}(\mathbf{W}) = -\frac{1}{N}\mathrm{LL}(\mathbf{W}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{nc}\log\mu_{nc} \qquad \text{con} \quad \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n) \quad \text{y} \quad \boldsymbol{a}_n = \mathbf{W}^t\boldsymbol{x}_n$$

Ejemplo (cont.): neg-log-verosimilitud de $\mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$ con $\mathcal{D} = \{((1,0,0)^t, (1,0)^t), ((1,1,1)^t, (0,1)^t)\}$

$$\mathrm{NLL}(\mathbf{W}) = -\frac{1}{2}\mathrm{LL}(\mathbf{W}) = 0.1269$$

Riesgo empírico con log-pérdida: es el mismo que la NLL

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N}\ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) = \mathrm{NLL}(\mathbf{W}) \quad \text{con} \quad \ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) = -\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n) = -\sum_{c=1}^{C} y_{nc}\log\mu_{nc}$$

- Si el modelo asigna probabilidad uno en la clase correcta, la pérdida es nula;
- Si no, la pérdida será positiva y será más grande cuando menor sea la probabilidad asignada en la clase correcta

Aprendizaje por mínima NLL: aprendizaje por máxima verosimilitud planteado como un problema de minimización

$$\mathbf{W}^* = \operatorname*{argmin}_{\mathbf{W}} \ \mathrm{NLL}(\mathbf{W})$$

Ejemplo (cont.): $\mathcal{D} = \{((1,0,0)^t, (1,0)^t), ((1,1,1)^t, (0,1)^t)\}$; por simplicidad, suponemos que hemos de elegir por mínima NLL entre

$$\mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix} \qquad \text{y} \qquad \tilde{\mathbf{W}}^t = \begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

Elegimos $\mathbf{W}$ ya que su NLL, 0.1269 (calculada antes), es menor que la de $\tilde{\mathbf{W}}$:

$$\text{NLL}(\tilde{\mathbf{W}}) = -\frac{1}{2}(\log \tilde{\mu}_{11} + \log \tilde{\mu}_{22}) = -\log \frac{e^{-1}}{e^{-1} + e^{1}} = \log(1 + e^2) = 2.1269$$

5 Algoritmo de aprendizaje con descenso por gradiente

Propósito: a diferencia del riesgo con pérdida 01 (tasa de error en entrenamiento), el riesgo con log-pérdida (NLL) es derivable, por lo cual podemos minimizarlo con técnicas de optimización estándar como por ejemplo descenso por gradiente

Descenso por gradiente

Descenso por gradiente: algoritmo iterativo para minimizar un objetivo $\mathcal{L}(\boldsymbol{\theta})$ a partir de uno $\boldsymbol{\theta}_0$ dado

$$\boldsymbol{\theta}_{y+1} = \boldsymbol{\theta}_y - \eta_y \boldsymbol{\nabla} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_y} \qquad y = 0, 1, \ldots$$

Factor de aprendizaje: $\eta_y > 0$ juega el mismo papel que en Perceptrón; podemos elegir un valor constante pequeño, $\eta_y = \eta$

Dirección de descenso más pronunciada: $-\boldsymbol{\nabla} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_y}$ es el neg-gradiente del objetivo evaluado a $\boldsymbol{\theta}_y$

Convergència: si η no es muy grande y el objetivo es convexo (con forma de bol), converge a un mínimo (global)

Ejemplo: $\mathcal{L}(\theta) = \theta^2$, $\theta_0 = 10$, $\eta_t = 0.2$, $\frac{d\mathcal{L}}{d\theta} = 2\theta$ y tolerancia 0.01

In [3]:
```
import numpy as np
grad, theta, eta, tol, delta = lambda t: 2*t, 10.0, 0.2, 0.01, np.inf
print(np.round(delta, 4), np.round(theta, 4), np.round(theta*theta, 4))
while np.abs(delta) > tol:
        delta = -eta * grad(theta)
        theta += delta
        print(np.round(delta, 4), np.round(theta, 4), np.round(theta*theta, 4))
```

```
inf 10.0 100.0
-4.0 6.0 36.0
-2.4 3.6 12.96
-1.44 2.16 4.6656
-0.864 1.296 1.6796
-0.5184 0.7776 0.6047
-0.311 0.4666 0.2177
-0.1866 0.2799 0.0784
-0.112 0.168 0.0282
-0.0672 0.1008 0.0102
-0.0403 0.0605 0.0037
-0.0242 0.0363 0.0013
-0.0145 0.0218 0.0005
-0.0087 0.0131 0.0002
```

Descenso por gradiente aplicado a regresión logística

NLL: la NLL es una función objetivo convexa

$$\mathrm{NLL}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N} -\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n) \qquad \text{con} \quad \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n) \quad \text{y} \quad \boldsymbol{a}_n = \mathbf{W}^t \boldsymbol{x}_n$$

Gradiente de la NLL: haremos uso del siguiente resultado, sin demostración

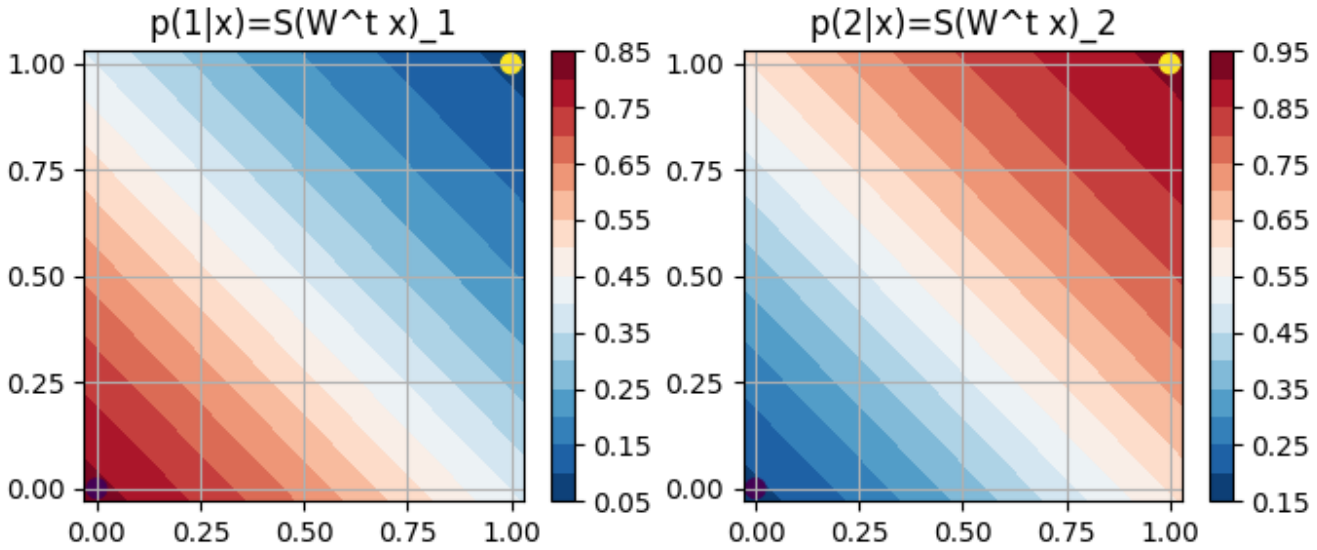
$$\begin{pmatrix} \frac{\partial \mathrm{NLL}}{\partial W_{01}} & \cdots & \frac{\partial \mathrm{NLL}}{\partial W_{0C}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{NLL}}{\partial W_{D1}} & \cdots & \frac{\partial \mathrm{NLL}}{\partial W_{DC}} \end{pmatrix} = \frac{\partial \mathrm{NLL}}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial(-\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n))}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n(\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t$$

Descenso por gradiente aplicado a regresión logística: $\mathbf{W}_0 = \mathbf{0}; \quad \mathbf{W}_{i+1} = \mathbf{W}_i - \eta_i \frac{\partial \mathrm{NLL}}{\partial \mathbf{W}^t}\Big|_{\mathbf{W}_i} \quad i = 0, 1, \ldots$

In [31]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[1, 0, 0], [1, 1, 1]]).astype(float); N, D = X.shape
y = np.array([[1, 0], [0, 1]]).astype(int); _, C = y.shape
W = np.zeros((D, C)).astype(float); Z = np.zeros((N, C)).astype(float)
eta, tol, delta = 0.2, 0.01, np.inf
while np.any(np.abs(delta) > tol):
    Z = np.apply_along_axis(np.exp, 1, X @ W)
    Z = Z/Z.sum(axis=1, keepdims=True); Z -= y
    delta = -eta/N * np.einsum('ij,ik->jk', X, Z)
    W += delta
print(W.T)
```

```
[[ 0.7297801 -0.9399284 -0.9399284]
 [-0.7297801  0.9399284  0.9399284]]
```

In [32]:
```
x1, x2 = np.meshgrid(np.linspace(-.03, 1.03, 50), np.linspace(-.03, 1.03, 50))
XX = np.c_[np.ones(50*50), np.ravel(x1), np.ravel(x2)]
Z = np.apply_along_axis(np.exp, 1, XX @ W); Z = Z/Z.sum(axis=1, keepdims=True)
_, axs = plt.subplots(1, 2, figsize=(8, 3))
for i, ax in enumerate(axs.flat):
    ax.set_xticks(np.linspace(0., 1, 5)); ax.set_yticks(np.linspace(0., 1, 5));
    ax.grid(); ax.set_title(f'p({i+1}|x)=S(W^t x)_{i+1}')
    cp = ax.contourf(x1, x2, Z[:, i].reshape(x1.shape), 15, cmap='RdBu_r')
    plt.colorbar(cp, ax=ax); ax.scatter(*X[:,1:].T, c=range(1,C+1), s=50);
```



Ejercicios T2.4 Regresión logística

Cuestión: Sea un modelo de regresión logística en notación compacta (homogénea) para un problema de clasificación en $C = 3$ clases y datos representados mediante vectores de dimensión $D = 2$

$$p(\boldsymbol{y} \mid \boldsymbol{x}; \mathbf{W}) = \mathrm{Cat}(\boldsymbol{y} \mid \mathcal{S}(\mathbf{W}^t \boldsymbol{x})) \quad \text{con} \quad \mathbf{W}^t = \begin{pmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \in \mathbb{R}^{C \times (D+1)}$$

La probabilidad P de que $\boldsymbol{x} = (1, 0.5, 0.5)^t$ pertenezca a la clase 1 és:

1. $P < 0.25$
2. $0.25 \leq P < 0.5$
3. $0.5 \leq P < 0.75$
4. $0.75 \leq P$

Solución: la 3

$$p(y = 1 \mid \boldsymbol{x}; \mathbf{W}) = \mathcal{S}\left(\begin{pmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0.5 \\ 0.5 \end{pmatrix} \right)_1 = \mathcal{S}((1, 0, 0)^t)_1 = \frac{e}{e+2} = \frac{1}{1 + 2/e} = 0.5761$$

Problema: Sea un modelo de regresión logística en notación compacta (homogénea) para un problema de clasificación en $C = 3$ clases y datos representados mediante vectores de dimensión $D = 2$

$$p(\boldsymbol{y} \mid \boldsymbol{x}; \mathbf{W}) = \mathrm{Cat}(\boldsymbol{y} \mid \mathcal{S}(\mathbf{W}^t \boldsymbol{x})) \quad \text{con} \quad \mathbf{W}^t = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \in \mathbb{R}^{C \times (D+1)}$$

Actualiza el valor de $\mathbf{W}$ mediante una iteración de descenso por gradiente con conjunto de entrenamiento $\mathcal{D} = \{(\boldsymbol{x} = (1, 1, 1)^t, y = 1)\}$ y factor de aprendizaje $\eta = 0.1$.

Solución:

$$\boldsymbol{a} = \mathbf{W}^t \boldsymbol{x} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\boldsymbol{\mu} = S(\boldsymbol{a}) = \frac{1}{1+2e} \begin{pmatrix} 1 \\ e \\ e \end{pmatrix} = \begin{pmatrix} 0.1554 \\ 0.4223 \\ 0.4223 \end{pmatrix}$$

$$\begin{aligned} \mathbf{W} &= \mathbf{W} - \eta \, \boldsymbol{x}(\boldsymbol{\mu} - \boldsymbol{y})^t \\ &= \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} - 0.1 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (-0.8446, 0.4223, 0.4223) \\ &= \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} -0.0845 & 0.0422 & 0.0422 \\ -0.0845 & 0.0422 & 0.0422 \\ -0.0845 & 0.0422 & 0.0422 \end{pmatrix} \\ &= \begin{pmatrix} 1.0845 & -0.0422 & 0.9578 \\ -0.9155 & 0.9578 & -1.0422 \\ 0.0845 & -0.0422 & 0.9578 \end{pmatrix} \end{aligned}$$

Problema: La siguiente tabla presenta un conjunto de 2 muestras de entrenamiento $(n = \{1, 2\})$ de 2 dimensiones (x_{n1}, x_{n2}) procedentes de 2 clases $(c_n = \{1, 2\})$:

n	x_{n1}	x_{n2}	c_n
1	1	0	1
2	1	1	2

Adicionalmente, la siguiente tabla representa una matriz de pesos iniciales $\mathbf{W}$ con los pesos de cada una de las clases por columnas (en notación homogénea):

w_1	w_2
0	0
0	0
-0.25	0.25

Se pide:

1. Calcula el vector de logits asociado a cada muestra de entrenamiento.
2. Aplica la función softmax al vector de logits de cada muestra de entrenamiento.
3. Clasifica cada una de las muestras de entrenamiento. En caso de empate, elige cualquier clase.
4. Calcula el gradiente de la función NLL en el punto de la matriz de pesos iniciales.
5. Actualiza la matriz de pesos iniciales aplicando descenso por gradiente con factor de aprendizaje $\eta = 1.0$

Solución:

0. En primer lugar, debemos tener en cuenta que las muestras de entrenamiento se proporcionan por filas. Seguidamente tenemos que preparar las muestras de entrenamiento para que estén listas para ser utilizadas en regresión logística. Para ello, convertimos las muestras de entrenamiento a notación homogénea y las etiquetas de clase en codificación one-hot:

$$\boldsymbol{x_1} = (1,1,0)^t; \quad \boldsymbol{y_1} = (1,0)^t;$$

$$\boldsymbol{x_2} = (1,1,1)^t; \quad \boldsymbol{y_2} = (0,1)^t;$$

$$\mathbf{W}_0 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -0.25 & 0.25 \end{pmatrix}$$

1. Vector de logits para cada muestra de entrenamiento, esto es, $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$:

$$\boldsymbol{a_1} = \mathbf{W}_0^t \boldsymbol{x_1} = \begin{pmatrix} 0 & 0 & -0.25 \\ 0 & 0 & 0.25 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = (0,0)^t$$

$$\boldsymbol{a_2} = \mathbf{W}_0^t \boldsymbol{x_2} = \begin{pmatrix} 0 & 0 & -0.25 \\ 0 & 0 & 0.25 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = (-0.25, 0.25)^t$$

2. Aplicamos la función softmax para obtener $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$:

$$\boldsymbol{\mu}_1 = \mathcal{S}(\boldsymbol{a_1}) = \left(\frac{e^0}{e^0 + e^0}, \frac{e^0}{e^0 + e^0} \right)^t = (0.5, 0.5)^t$$

$$\boldsymbol{\mu}_2 = \mathcal{S}(\boldsymbol{a_2}) = \left(\frac{e^{-0.25}}{e^{-0.25} + e^{0.25}}, \frac{e^{0.25}}{e^{-0.25} + e^{0.25}} \right)^t = (0.38, 0.62)^t$$

3. Clasificación de $\boldsymbol{x_1}$ y $\boldsymbol{x_2}$:

$$\hat{c}(\boldsymbol{x_1}) = \underset{c}{\operatorname{argmax}}\ \mu_{1c} = \underset{c}{\operatorname{argmax}}\ [0.5, 0.5] = 1$$
$$\hat{c}(\boldsymbol{x_2}) = \underset{c}{\operatorname{argmax}}\ \mu_{2c} = \underset{c}{\operatorname{argmax}}\ [0.38, 0.62] = 2$$

4. Calculamos el gradiente de la función NLL en el punto correspondiente a la matriz inicial de pesos:

$$\begin{aligned}
\frac{\partial \mathrm{NLL}}{\partial \mathbf{W}^t}\Big|_{\mathbf{W}_0} &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n(\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t \\
&= \frac{1}{2}\cdot\left(\boldsymbol{x}_1(\boldsymbol{\mu}_1 - \boldsymbol{y}_1)^t + \boldsymbol{x}_2(\boldsymbol{\mu}_2 - \boldsymbol{y}_2)^t\right) \\
&= \frac{1}{2}\cdot\left(\begin{pmatrix}1\\1\\0\end{pmatrix}\cdot(-0.5, 0.5) + \begin{pmatrix}1\\1\\1\end{pmatrix}\cdot(0.38, -0.38)\right) \\
&= \frac{1}{2}\cdot\left(\begin{pmatrix}-0.5 & 0.5\\-0.5 & 0.5\\0 & 0\end{pmatrix} + \begin{pmatrix}0.38 & -0.38\\0.38 & -0.38\\0.38 & -0.38\end{pmatrix}\right) \\
&= \frac{1}{2}\cdot\begin{pmatrix}-0.12 & 0.12\\-0.12 & 0.12\\0.38 & -0.38\end{pmatrix} \\
&= \begin{pmatrix}-0.06 & 0.06\\-0.06 & 0.06\\0.19 & -0.19\end{pmatrix}
\end{aligned}$$

5. Actualizamos la matriz de pesos:

$$
\begin{aligned}
\mathbf{W}_1^t &= \mathbf{W}_0^t - \eta \frac{\partial \mathrm{NLL}}{\partial \mathbf{W}^t}\Big|_{\mathbf{w}_0} \\
&= \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -0.25 & 0.25 \end{pmatrix} - 1 \cdot \begin{pmatrix} -0.06 & 0.06 \\ -0.06 & 0.06 \\ 0.19 & -0.19 \end{pmatrix} \\
&= \begin{pmatrix} 0.06 & -0.06 \\ 0.06 & -0.06 \\ -0.44 & 0.44 \end{pmatrix}
\end{aligned}
$$