U2.1 Discriminant functions

Index

1 Discriminant functions

Notation:

- **Objects (inputs):** $\boldsymbol{x} \in \mathcal{X},$ typically vectors of D real features, $\mathcal{X} = \mathbb{R}^D,\ D \geq 1$
- **Class labels (outputs):** $y \in \mathcal{Y}$ or $c \in \mathcal{C},$ typically $\mathcal{Y} = \mathcal{C} = \{1, \ldots, C\},\ C \geq 2$

Classical representation of classifiers: with a **discriminant function** per class, $g_c(x),$ to measure the (pseudo-)probability of x belonging to c

$$c(\boldsymbol{x}) = \underset{c}{\operatorname{argmax}}\ g_c(\boldsymbol{x})$$

Example: classifier in 3 classes for $\boldsymbol{x} = (x_1, x_2)^t \in [0,1]^2$

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

Bayes decision rule: with $g_c(\boldsymbol{x}) = P(c \mid \boldsymbol{x})$ or $g_c(\boldsymbol{x}) = P(c)\, p(\boldsymbol{x} \mid c)$

2 Linear discriminant functions

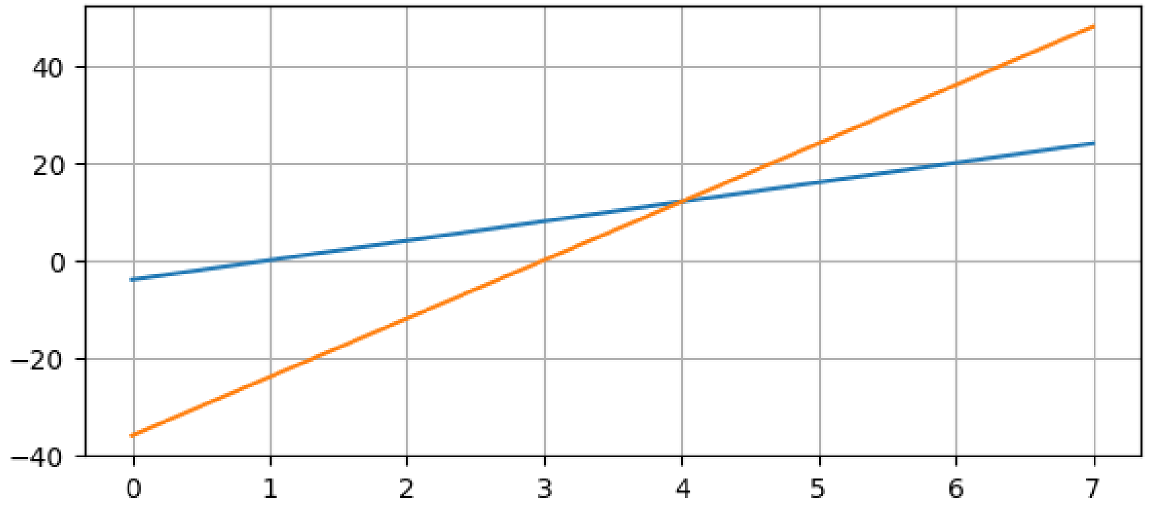
Linear discriminant function: linear function with the features

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

- $\boldsymbol{w}_c = (w_{c1}, w_{c2}, \dots, w_{cD})^t$ is the **vector of weights** of class c
- w_{c0} is the **threshold weight** of class c

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

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));
```



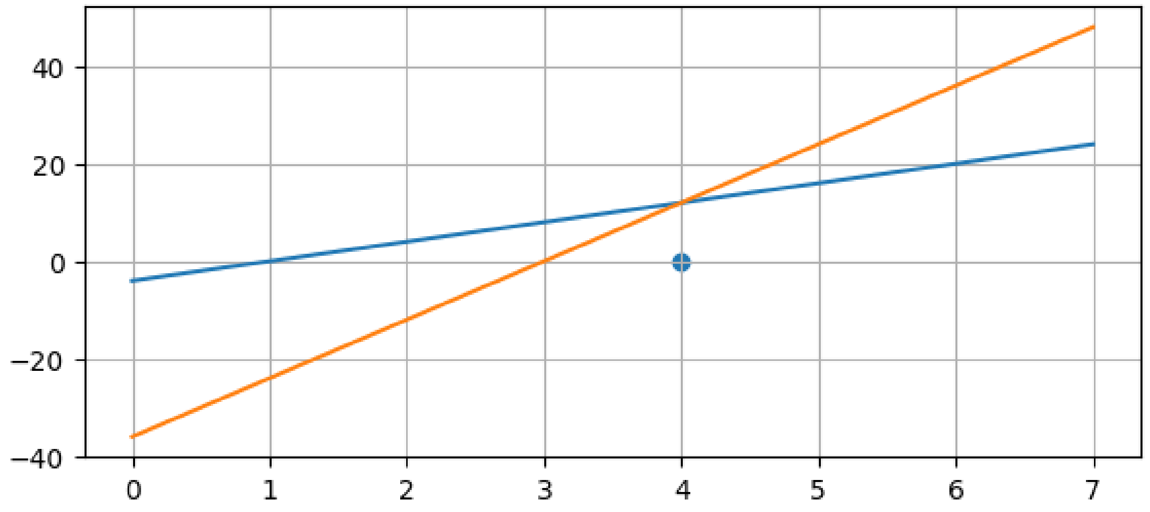
3 Decision boundaries

Decision boundary between classes: geometric location where the discriminants of two (or more) classes are equal to each other

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

Example (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);
```



Linear boundaries: linear discriminants originate linear boundaries of dimension $D-1$

- One point, if $\mathcal{X}=\mathbb{R}$
- A line (straight lines), if $\mathcal{X}=\mathbb{R}^2$
- A surface (planes), if $\mathcal{X}=\mathbb{R}^3$

Boundaries in general: **hypersurfaces** defined by the equations

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

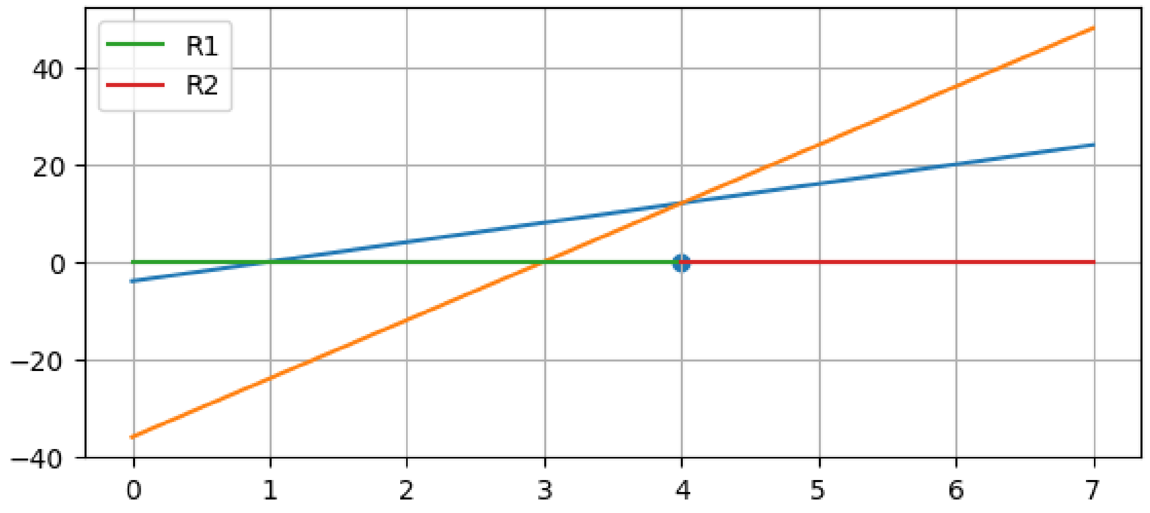
4 Decision regions

Decision region of a class: Geometric locations where the discriminant of the class wins over the rest

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

Example (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 Equivalent classifiers

Purpose: simplify a given classifier, $c(\boldsymbol{x}) = \operatorname{argmax}_c g_c(\boldsymbol{x})$

Equivalent classifier: $c'(\boldsymbol{x}) = \operatorname{argmax}_c g'_c(\boldsymbol{x})$ is equivalent to $c(\boldsymbol{x})$ if $c(\boldsymbol{x}) = c'(\boldsymbol{x})$ for all $\boldsymbol{x}$

Construction: if $f : \mathbb{R} \to \mathbb{R}$ is strictly increasing, the following classifier is equivalent to $c(\boldsymbol{x})$

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

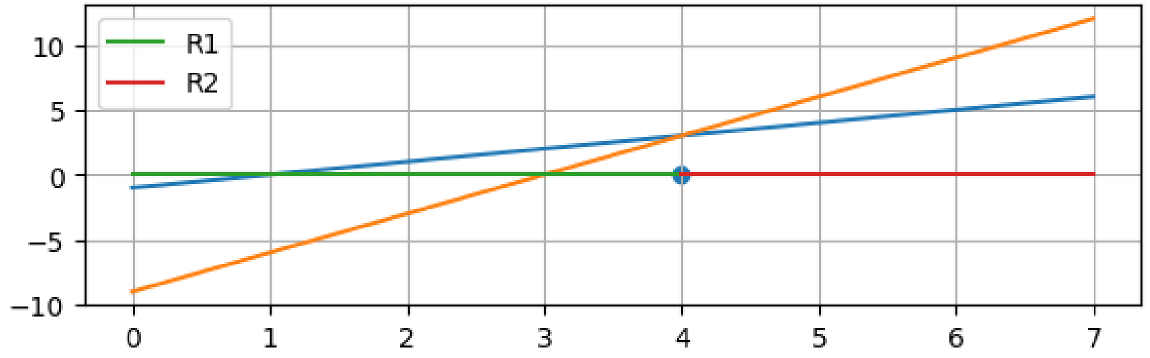
where $\operatorname{const}(\boldsymbol{x})$ is any function that may vary (or not) with $\boldsymbol{x}$, but not with c

Usual strictly increasing functions: to build simplified equivalent classifiers

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

Example (cont.): $g'_1(x) = x - 1,\ g'_2(x) = 3x - 9$ with $f(z) = \frac{1}{4}z$

In [2]:
```
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, 2)); 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();
```



U2.1 Discriminant functions

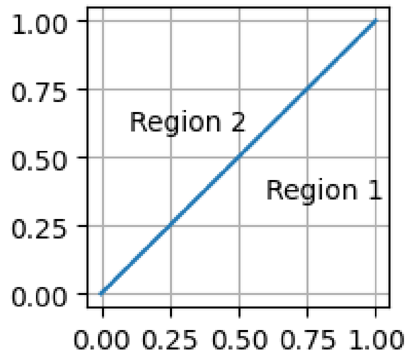
2023_01_26_Question_2: Given the classifier in 2 classes defined by their weight vectors $\boldsymbol{w}_1 = (-2, 3, 3)^t,\ \boldsymbol{w}_2 = (0, 2, -2)^t$ in homogeneous notation, which of the following sets of vectors **does not** define a classifier equivalent to the one given?

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

Solution: Option 4.

2023_01_17_Question 3: Let the classifier in two classes be defined by its boundary and decision regions in the figure:

```
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, 'Region 2'); plt.text(0.6, 0.35, 'Region 1');
```



Which of the following weight vectors (in homogeneous notation) defines a classifier equivalent to that given?

1. $\boldsymbol{w}_1 = (0, -2, 0)^t$ and $\boldsymbol{w}_2 = (0, 0, -2)^t$.
2. $\boldsymbol{w}_1 = (0, 2, 0)^t$ and $\boldsymbol{w}_2 = (0, 0, 2)^t$.
3. $\boldsymbol{w}_1 = (0, 0, 2)^t$ and $\boldsymbol{w}_2 = (0, 2, 0)^t$.
4. All the above weight vectors define equivalent classifiers.

Solution: Option 2.

2022_02_03_Question: Let $\boldsymbol{x}$ be an object to be classified into a class of C possibles. Indicate which of the following classifiers **is not** minimum error (or choose the last option if all three are minimum error):

1. $c(\boldsymbol{x}) = \operatorname*{argmax}_{c=1,\dots,C} \log p(c) + \log p(\boldsymbol{x}|c) - \log p(\boldsymbol{x})$
2. $c(\boldsymbol{x}) = \operatorname*{argmax}_{c=1,\dots,C} -\log p(\boldsymbol{x}, c)$
3. $c(\boldsymbol{x}) = \operatorname*{argmax}_{c=1,\dots,C} \log p(c) + \log p(\boldsymbol{x}|c)$
4. The three classifiers above are of minimum error

Solution: Option 2.

Problem (cont. of problem T1.2): Consider the classification of Iris flowers as setosa or non-setosa based on petal length, x. It is known that the distributions of x for setosa and non-setosa can be approximated by normal distributions of means and standard deviations:

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

Likewise, it is known that the a priori probability of setosa is $1/3$ and that the a posteriori probability that a flower of petal length 2 is setosa is 0.89. Find the decision boundary between setosa and non-setosa, and determine the decision regions of the classes.

Solution:

Decision boundary between setosa and non-setosa classes:

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

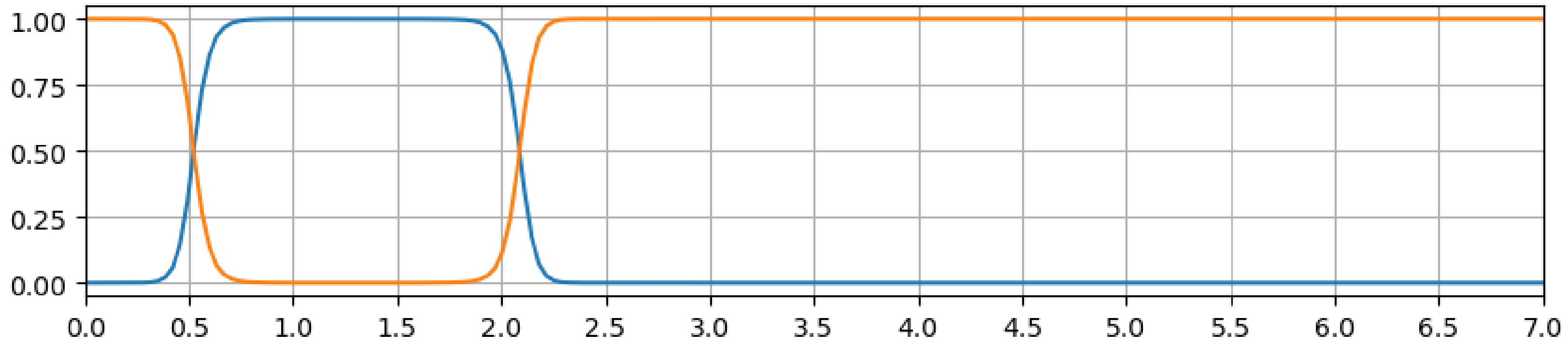
Decision regions:

- Normal densities are defined throughout $\mathbb{R}$, but we are only interested in $\mathbb{R}^+$
- The boundary found divides $\mathbb{R}^+$ into three intervals: $(0, 0.53)$, $(0.53, 2.09)$ and $(2.09, \infty)$
- Since $x = 2$ is setosa, $\mathcal{R}_{\text{setosa}} = (0.53, 2.09)$ and $\mathcal{R}_{\text{not-setosa}} = (0, 0.53) \cup (2.09, \infty)$

Since the normal distribution is positive throughout $\mathbb{R}$, the assumption of normality of the conditional densities implies that we assume any (positive) length of petals, both setosa and non-setosa, is possible. Obviously, it is practically impossible to find setosa or non-setosa petal length in $(0, 0.53)$. Now, with the assumption made, it is highly improbable but not theoretically impossible. Thus, since the conditional of non-setosa is flatter than that of setosa, the calculations indicate that in $(0, 0.53)$ the flower is more likely to be non-setosa.

We check the result with graphs of the posterior probabilities as a function of x:

```
In [2]: 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));
```



U2.2 Perceptron Algorithm

Index

1 Perceptron algorithm

Source: first machine learning algorithm; proposed by Frank Rosenblatt in 1958

Input: training dataset, $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$, with $\boldsymbol{x}_n \in \mathbb{R}^D$ and $y_n \in \{1, \dots, C\}$ for all n

Output: weights of the linear classifier, $c(\boldsymbol{x}) = \operatorname{argmax}_c\ g_c(\boldsymbol{x})$, with $g_c(\boldsymbol{x}) = \boldsymbol{w}_c^t\, \boldsymbol{x} + w_{c0}$ for all c

Homogeneous or compact notation: $\boldsymbol{x} = (1, x_1, \dots, x_D)^t$ and $\boldsymbol{w}_c = (w_{c0}, w_{c1}, \dots, w_{cD})^t$; so, $g_c(\boldsymbol{x}) = \boldsymbol{w}_c^t\, \boldsymbol{x}$

Objective: minimise the number of training errors

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

Objective with margin $b \geq 0$: extended objective ($b = 0$) for better generalisation

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

Interpretation: the pseudo-probability of belonging to the correct class has to be higher than that of any other class at least by a margin b

Perceptron algorithm: basic version with **learning rate** $\alpha > 0$ to control the learning speed

```
Input: data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$   weights $\{\boldsymbol{w}_c\}$   learning rate $\alpha \in \mathbb{R}^{>0}$   margin $b \in \mathbb{R}^{\geq 0}$
Output: optimised weights $\{\boldsymbol{w}_c\}$
repeat
    for all training sample $\boldsymbol{x}_n$
        err = False
        for all class $c \neq y_n$
            if $\boldsymbol{w}_c^t \boldsymbol{x}_n + b > \boldsymbol{w}_{y_n}^t \boldsymbol{x}_n$ :   $\boldsymbol{w}_c = \boldsymbol{w}_c - \alpha \boldsymbol{x}_n;$   err = True
        if err:   $\boldsymbol{w}_{y_n} = \boldsymbol{w}_{y_n} + \alpha \boldsymbol{x}_n$
until no training sample is misclassified
```

Implementation: function for simple learning problems

```
In [1]: import numpy as np
        def perceptro(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 Example

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

Trace:

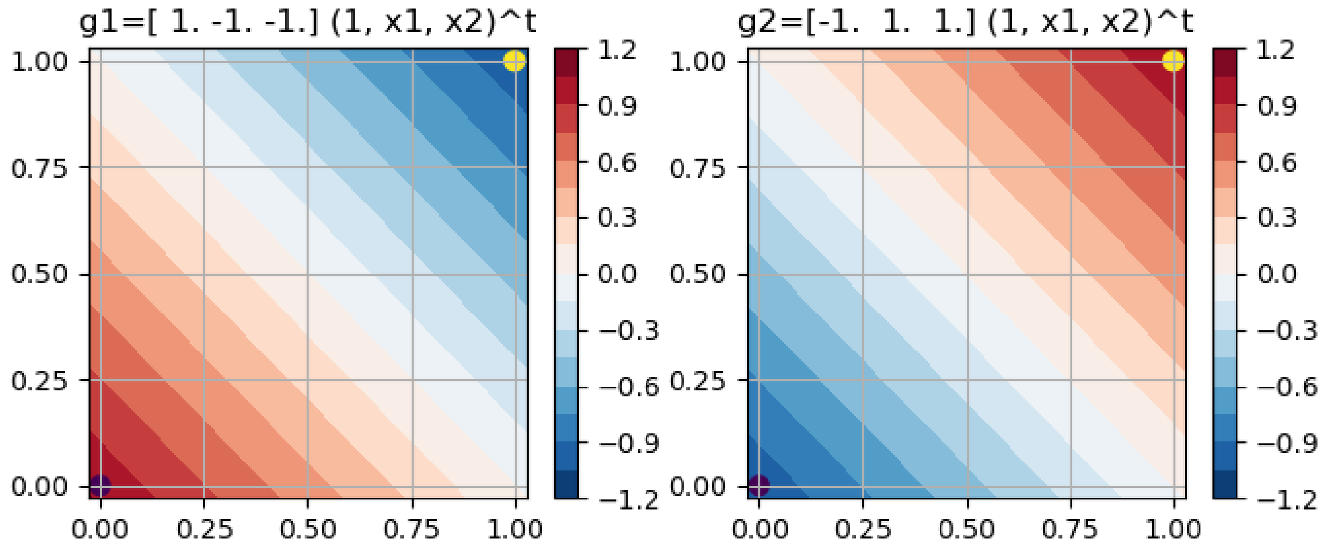
Iteration	n	$\boldsymbol{w}_{y_n}^t \boldsymbol{x}_n$	$c, c \neq y_n$	$\boldsymbol{w}_c^t \boldsymbol{x}_n + b$	$\boldsymbol{w}_1^t$	$\boldsymbol{w}_2^t$
					$(0,0,0)$	$(0,0,0)$
1	1	$(0,0,0)(1,0,0)^t = 0$	2	$(0,0,0)(1,0,0)^t + 0.1 = 0.1$		$(-1,0,0)$
1	1				$(1,0,0)$	
1	2	$(-1,0,0)(1,1,1)^t = -1$	1	$(1,0,0)(1,1,1)^t + 0.1 = 1.1$	$(0,-1,-1)$	
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]:
```python
X = np.array([[0, 0], [1, 1]]); y = np.array([0, 1], dtype=int);
W, E, k = perceptro(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
```

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

In [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)
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}={Wt[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 Convergence and quality of the solution

Convergence: Perceptron converges if training samples are **linearly separables**

Effect of learning rate $\alpha > 0$: it converges independently from the selected value, although slowly if α is very small

Effect of margin $b \geq 0$: it converges with centered boundaries if value is selected close to the maximum margin that allows to linearly classify the training samples with margin; if the margin is too large, Perceptron does not converge

- It is necessary to carry out experiments with different values for the margin to find that value that generalises optimally

U2.2 Perceptron algorithm

2023_01_26_Problem: The following table includes a set of 4 two-dimensional learning samples from 3 classes, $c = 1, 2, 3$.

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

It is requested:

1. (1.5 points) Make an execution trace of an iteration of the Perceptron algorithm, with learning factor $\alpha = 1$ margin $b = 0.1$ and the following initial weights of each class by columns:

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

2. (0.5 points) Classify the test sample $\boldsymbol{x} = (5, 5)^t$ using a linear classifier with the weight vectors obtained in the previous section.

Solution:

1. An iteration of the Perceptron algorithm with one misclassified sample and resulting weights:

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

2. Classification of the test sample: $g_1(\boldsymbol{x}) = -31 \quad g_2(\boldsymbol{x}) = -24 \quad g_3(\boldsymbol{x}) = -49 \quad \Rightarrow \quad c(\boldsymbol{x}) = 2$

2023_01_17_Question 4: Suppose we are applying the Perceptron algorithm, with learning factor $\alpha = 1$ and margin $b = 0.1$, to a set of 4 two-dimensional learning samples for a 4 class problem, $c = 1, 2, 3, 4$. At a given moment in the execution of the algorithm, the weight vectors $\boldsymbol{w}_1 = (-2, -2, -6)^t$, $\boldsymbol{w}_2 = (-2, -2, -6)^t$, $\boldsymbol{w}_3 = (-2, -4, -4)^t$, $\boldsymbol{w}_4 = (-2, -4, -4)^t$. Assuming that the sample $(\boldsymbol{x}, c) = ((4, 5)^t, 2)$ is to be processed next, how many weight vectors will be modified?

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

Solution: Option 4.

2022_01_27_Problem: In the table on the left, a set of 3 two-dimensional learning samples of 3 classes is provided, while in the table on the right, a set of initial weights is provided for each class

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

It is requested:

1. (1.5 points) Perform an execution trace of one iteration of the Perceptron algorithm, with learning factor $\alpha = 1$, margin $\gamma = 0.1$, and using the initial weights provided.
2. (0.5 points) Graphically represent the decision regions of the resulting classifier, as well as the decision boundaries needed for their representation.

Solution: A Perceptron iteration with one misclassified sample yields the following final weights:

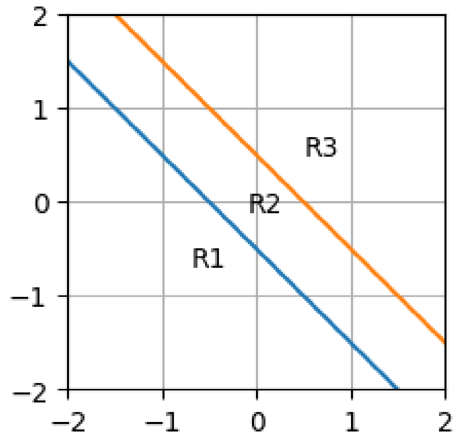
	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$
w_{c0}	-1	0	-2
w_{c1}	-2	0	4
w_{c2}	-2	0	4

Boundaries:

$$\begin{aligned}
\boldsymbol{w}_1^t\boldsymbol{x} = \boldsymbol{w}_2^t\boldsymbol{x} &\Rightarrow -1 - 2x_1 - 2x_2 = 0 \Rightarrow x_2 = -x_1 - 0.5\\
\boldsymbol{w}_1^t\boldsymbol{x} = \boldsymbol{w}_3^t\boldsymbol{x} &\Rightarrow -1 - 2x_1 - 2x_2 = -2 + 4x_1 + 4x_2 \Rightarrow x_2 = -x_1 + 1/6\\
\boldsymbol{w}_2^t\boldsymbol{x} = \boldsymbol{w}_3^t\boldsymbol{x} &\Rightarrow 0 = -2 + 4x_1 + 4x_2 \Rightarrow x_2 = -x_1 + 0.5
\end{aligned}$$

The graphical representation of the three decision regions with the two decision boundaries involved is as follows:

```
In [2]: import numpy as np; import matplotlib.pyplot as plt
        fig = plt.figure(figsize=(2.5, 2.5)); 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)); plt.plot((-2, 2), (2.5, -1.5));
        plt.text(-.7, -.7, 'R1'); plt.text(-.1, -.1, 'R2'); plt.text(.5, .5, 'R3');
```



2022_01_13_Question 3: Suppose we are applying the Perceptron algorithm, with learning factor $\alpha = 1$ and margin $b = 0.1$, to a set of 3 two-dimensional learning samples for a problem of 2 classes. It is known that, after processing the first 2 samples, the weight vectors $\boldsymbol{w}_1 = (0, 0, 1)^t$, have been obtained $\boldsymbol{w}_2 = (0, 0, -1)^t$. Likewise, it is known that, after processing the last sample, $(\boldsymbol{x}_3, c_3)$, the weight vectors $\boldsymbol{w}_1 = (-1, -5, -2)^t,$ and $\boldsymbol{w}_2 = (1, 5, 2)^t$. Which of the following samples is that last sample?

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

Solution: Option 3.

U2.3 Error estimation

Index

1 Theoretical error of a classifier

Purpose: determine the error probability of a given classifier, $c(\boldsymbol{x}) = \operatorname{argmax}_c g_c(\boldsymbol{x})$

A posteriori error: error probability for a given $\boldsymbol{x}$; one minus the probability of classifying $\boldsymbol{x}$ correctly

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

(A priori) error with discrete $\mathcal{X}$: expected error probability according to the **unconditional probability** of $\boldsymbol{x}$

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

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

Theoretical knowledge

x_1	x_2	$P(\boldsymbol{x})$	$P(1 \mid \boldsymbol{x})$	$P(2 \mid \boldsymbol{x})$
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

Theoretical error of a given classifier

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

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

(A priori) error with continuous $\mathcal{X}$: expected error probability according to the **unconditional probability density** of $\boldsymbol{x}$

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

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

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

Theoretical error of the given classifier:

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

2 Error of Bayes' classifier

Bayes' classifier or MAP decision rule (maximum a posteriori): $c^*(\boldsymbol{x}) = \operatorname{argmax}_c \ P(c \mid \boldsymbol{x})$

A posteriori Bayes' error: $\varepsilon(c^*(\boldsymbol{x})) = 1 - P(c^*(\boldsymbol{x}) \mid \boldsymbol{x}) = 1 - \max_c P(c \mid \boldsymbol{x})$

(A priori) Bayes' error with $\mathcal{X}$ discrete: $\varepsilon^* = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c^*(\boldsymbol{x}))] = \sum_{\boldsymbol{x}\in\mathcal{X}} P(\boldsymbol{x})\,\varepsilon(c^*(\boldsymbol{x}))$

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

Theoretical knowledge

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

Bayes' error

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

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

(A priori) Bayes' error with continuous $\mathcal{X}$: $\varepsilon^* = \mathbb{E}_{\boldsymbol{x}}[\varepsilon(c^*(\boldsymbol{x}))] = \int p(\boldsymbol{x})\,\varepsilon(c^*(\boldsymbol{x}))\,d\boldsymbol{x}$

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

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

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

3 Error estimation of a classifier

Impossibility to calculate the exact theoretical error: since we do not have access to the theoretical knowledge

Error estimation by resubstitution: consists of using the error rate computed on the training data

Resubtitution is optimistic: a classifier that is limited to classifying the training data well (e.g. via memorization) will yield an error rate very close to zero, but it will perform poorly on future data, not seen in the training set

Error estimation by holdout: consists of "leaving out" of the training set a part of the available data, which we call the **test set,** and compute the error rate on the test set

Holdout is (slightly) pessimistic: since it estimates the error of a classifier trained with less data than available, and in general the error rate increases as the number of training samples is reduced

Normal approximation to the distribution of the holdout estimator: if the number of test samples, M, is large, the holdout estimator can be considered a normal random variable whose mean is the theoretical error we want to estimate and the variance is inversely proportional to M

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

95% Confidence Interval: the normal approximation allows to provide the error estimation with a 95% confidence interval

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

Example: if we have 100 errors when classifying $M = 2000$ test samples

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

U2.3 Error estimation

2023_01_17_Question 2: Let be a four-class classification problem for data of the type $\boldsymbol{x} = (x_1, x_2)^t \in \{0,1\}^2$, with the probability distributions from the table:

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

Indicate in which interval the Bayes' error, ε^*, is:

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

Solution: $\varepsilon^* = 0.65$

2022_01_27_Question 3: Let be a three-class classification problem for data of the type $\boldsymbol{x} = (x_1, x_2)^t \in \{0,1\}^2$, with the probability distributions from the table:

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

Indicate in which interval the error of the classifier $c(\boldsymbol{x})$ given in the table, ε, is:

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

Solution: $\varepsilon = 0.70$

2022_01_13_Question 2: The error probability of a classifier is estimated to be 20%. Determine what the minimum number of test samples, M, is necessary to achieve that the confidence interval at 95% of said error does not exceed $\pm 1\%$; that is, $I = [19\%, 21\%]$:

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

Solution: $M = 6147$

U2.4 Logistic regression

Index

1 One-hot encoding and categorical distribution

Categorical variable: random variable that takes a value from a finite set of (unordered) categories

Examples of categorical variables: RGB color, **class label,** vocabulary word, etc.

One-hot encoding: of a categorical variable y that takes a value among C possible, $\{1, \dots, C\}$

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

Categorical distribution: distribution of probabilities among the C possible categories of a categorical variable, whose probabilities are given by a parameter vector $\boldsymbol{\theta} \in [0,1]^C$ such that $\sum_c \theta_c = 1$

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

Convention: $0^0 = 1$ and $0 \log 0 = 0$; for example, with
$\boldsymbol{\theta} = (0.5, 0.5, 0)^t$, $\text{Cat}(\boldsymbol{y} = (1,0,0)^t \mid \boldsymbol{\theta}) = 0.5^1 0.5^0 0^0 = 0.5$

2 Probabilistic classification model with softmax

Probabilistic normalization of classifiers: every classifier defined with general discriminant functions can be represented by an equivalent classifier with probabilistically normalized discriminant functions

$$\begin{aligned} c(\boldsymbol{x}) &= \operatorname*{argmax}_c \; a_c && \text{where } a_c \text{ is the discriminant of class } c \text{ evaluated at } \boldsymbol{x} \\ &= \operatorname*{argmax}_c \; e^{a_c} && \text{with } h(z) = e^z \in \mathbb{R}^{\geq 0} \text{ strictly increasing} \\ &= \operatorname*{argmax}_c \; \frac{e^{a_c}}{\sum_{c'} e^{a_{c'}}} && \text{with } h(z) = kz,\, k \text{ positive constant (invariant with } c) \end{aligned}$$

The softmax function: transforms a vector of **logits** (non-normalized log-likelihoods) $\boldsymbol{a} \in \mathbb{R}^C$ into a vector of probabilities $[0,1]^C$

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

Probabilistic classification model with softmax: instead of predicting a single most likely class, we predict the probabilities of all classes from a logit predictor function, $f : \mathcal{X} \to \mathbb{R}^C$, governed by a parameter vector $\boldsymbol{\theta}$

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

Convenience of the model in inference: the prediction of the probabilities of all classes allows to apply more general rules than the MAP, for example in case of errors with different costs; also, if we want to apply the MAP rule, we do not need to softmax-normalize logits

Convenience of the model in learning: allows learning to be considered probabilistically, with standard criteria such as maximum likelihood; moreover, thanks to the softmax, $f(\boldsymbol{x}; \boldsymbol{\theta})$ can be freely chosen since it is not subject to probabilistic restrictions

3 Logistic regression

Logistic regression: model with softmax and **linear logits** with input (in homogeneous notation)

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

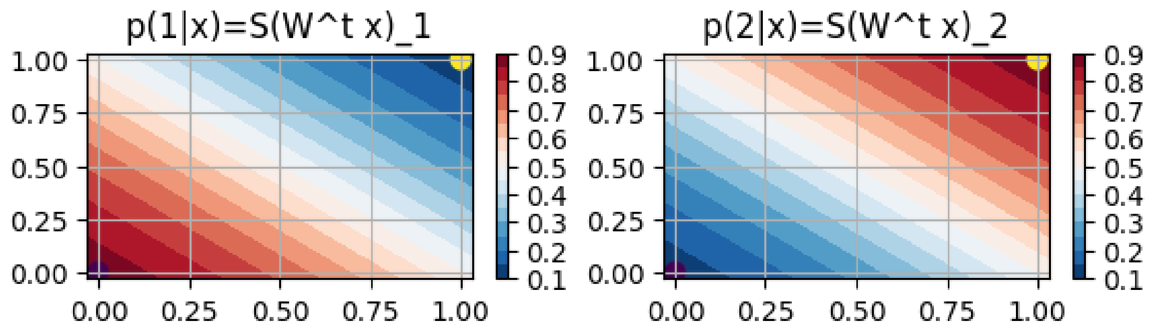
Difference with linear discriminant based classifiers: none, except that now we predict all class probabilities

Example (perceptron cont.): $C = D = 2, \quad a_1 = g_1(x_1, x_2) = -x_1 - x_2 + 1, \quad a_2 = g_2(x_1, x_2) = x_1 + x_2 - 1$

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

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

```
In [17]: 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=(7.0, 1.5))
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 Learning by maximum likelihood

Purpose: establish a criterion for learning $\mathbf{W}$ from a training dataset, $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$

Learning by maximum likelihood

Log-likelihood (conditional): log-likelihood of $\mathcal{D}$ interpreted as a function of $\mathbf{W}$

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

Example (cont.): log-likelihood of $\mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$ with $\mathcal{D} = \{((1,0,0)^t, (1,0)^t), ((1,1,1)^t, (0,1)^t)\}$

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

Learning by maximum likelihood: choose a $\mathbf{W}$ that gives maximum likelihood to $\mathcal{D}$

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

Considered as a minimization problem

Neg-log-likelihood: log-likelihood with the sign changed and normalized by the number of data samples

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

Example (cont.): neg-log-likelihood of $\mathbf{W}^t = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}$ with $\mathcal{D} = \{((1,0,0)^t,(1,0)^t),((1,1,1)^t,(0,1)^t)\}$

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

Empirical risk with log-loss: is the same as the NLL

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

- If the model assigns probability one to the correct class, the loss is zero;
- If not, the loss will be positive and will be greater when the probability assigned to the correct class is lower

Learning by minimum NLL: learning by maximum likelihood posed as a minimization problem

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

Example (cont.): $\mathcal{D} = \{((1,0,0)^t, (1,0)^t), ((1,1,1)^t, (0,1)^t)\};$ for simplicity, suppose we have to choose for minimum NLL between

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

We choose $\mathbf{W}$ since its NLL, 0.1269 (calculated before), is smaller than that of $\tilde{\mathbf{W}}$:

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

5 Learning algorithm with gradient descent

Purpose: unlike empirical risk with loss 01 (training error rate), empirical risk with log-loss (NLL) is differentiable, so we can minimize it with standard optimization techniques such as gradient descent

Gradient descent

Gradient descent: iterative algorithm to minimize an objective $\mathcal{L}(\boldsymbol{\theta})$ from a $\boldsymbol{\theta}_0$ given

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i \boldsymbol{\nabla}\mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_i} \qquad i = 0, 1, \dots$$

Learning factor: $\eta_i > 0$ plays the same role as Perceptron; we can choose a small constant value, $\eta_i = \eta$

Direction of steepest descent: $-\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_i}$ is the neg -gradient of the objective evaluated at $\boldsymbol{\theta}_i$

Convergence: if η is not very large and the objective is convex (bowl-shaped), it converges to a (global) minimum

Example: $\mathcal{L}(\theta) = \theta^2,\ \theta_0 = 9,\ \eta_t = 0.2,\ \frac{d\mathcal{L}}{d\theta} = 2\theta$ and tolerance 0.01

In [3]:
```python
import math
oL = math.inf
L, grad, theta, eta, tol, delta = lambda t: t**2, lambda t: 2*t, 9.0, 0.2, 0.01, -math.inf
print('theta delta     L')
print('----- ----- -----')
print(f'{theta:5.2f} {delta:5.2f} {L(theta):5.2f}')
while abs(delta) > tol:
        oL = L(theta)
        delta = -eta * grad(theta)
        theta += delta
        print(f'{theta:5.2f} {delta:5.2f} {L(theta):5.2f}')
```

```
theta delta     L
----- ----- -----
 9.00  -inf 81.00
 5.40 -3.60 29.16
 3.24 -2.16 10.50
 1.94 -1.30  3.78
 1.17 -0.78  1.36
 0.70 -0.47  0.49
 0.42 -0.28  0.18
 0.25 -0.17  0.06
 0.15 -0.10  0.02
 0.09 -0.06  0.01
 0.05 -0.04  0.00
 0.03 -0.02  0.00
 0.02 -0.01  0.00
 0.01 -0.01  0.00
```

Gradient descent applied to logistic regression

NLL: the NLL is a convex objective function

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

NLL gradient: we will use the following result, without proof

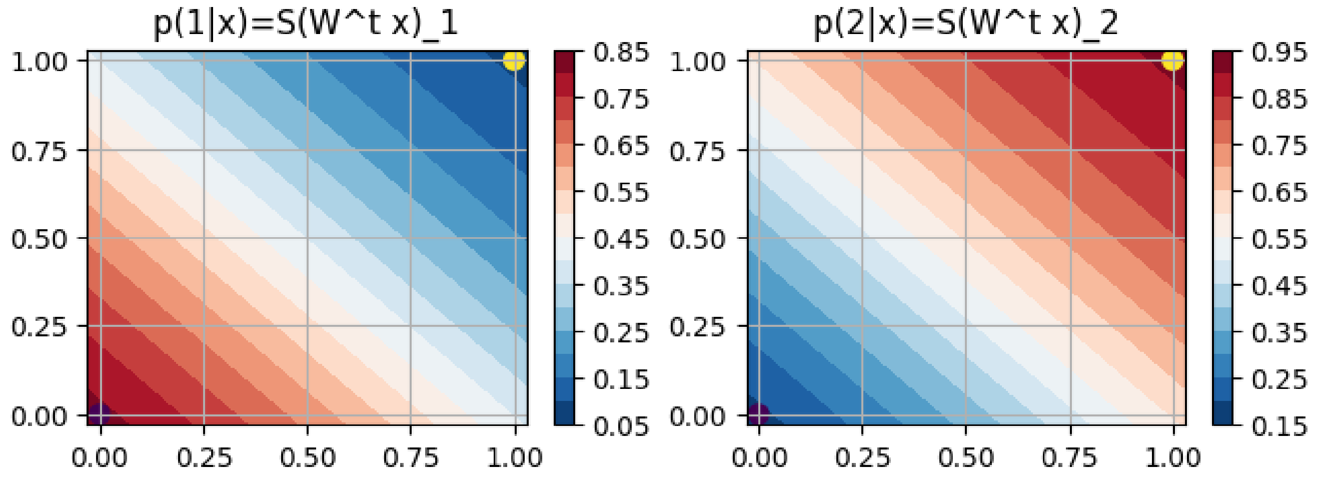
$$\begin{pmatrix} \frac{\partial\,\mathrm{NLL}}{\partial W_{11}} & \cdots & \frac{\partial\,\mathrm{NLL}}{\partial W_{1C}} \\ \vdots & \ddots & \vdots \\ \frac{\partial\,\mathrm{NLL}}{\partial W_{D1}} & \cdots & \frac{\partial\,\mathrm{NLL}}{\partial W_{DC}} \end{pmatrix} = \frac{\partial\,\mathrm{NLL}}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial(-\log p(\boldsymbol{y}_n \mid \boldsymbol{\mu}_n))}{\partial \mathbf{W}^t} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n(\boldsymbol{\mu}_n - \boldsymbol{y}_n)^t$$

Gradient descent applied to logistic regression: $\mathbf{W}_0 = \mathbf{0}; \quad \mathbf{W}_{i+1} = \mathbf{W}_i - \eta_i \frac{\partial\,\mathrm{NLL}}{\partial \mathbf{W}^t}\Big|_{\mathbf{W}_i} \quad i = 0, 1, \ldots$

```
In [20]: 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):
             a = X @ W
             mu = np.exp(a)
             mu /= np.sum(mu, axis=1, keepdims=True);
             Z = mu - y
             grad = (X.T @ Z)/N
             delta = eta*grad
             W -= delta
         print(W)
```

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

```
In [21]: 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, 2.5))
         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);
```



U2.4 Logistic regression

Question: Let be a logistic regression model for a classification problem in $C = 3$ classes and data represented by vectors of dimension $D = 2$

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

The probability P that $\boldsymbol{x} = (0.5, 0.5)^t$ belongs to class 1 is:

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

Solution: Option 3

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

Problem: Let be a logistic regression model in compact (homogeneous) notation for a classification problem in $C = 3$ classes and data represented by vectors of dimension $D = 3$

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

Update the value of $\mathbf{W}$ by one gradient descent iteration with training set $\mathcal{D} = \{(\boldsymbol{x} = (1, 1, 1)^t, y = 1)\}$ and learning factor $\eta = 0.1$.

Solution:

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

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

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

Problem: The following table shows a training set of 2 samples with 2 dimensions that belong to 2 classes:

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

In addition, the following table represents an initial weight matrix with the weights of each class per columns:

w_1	w_2
0	0
0	0
-0.25	0.25

Answer the following questions:

1. Compute the vector of logits for each training sample.
2. Apply the softmax function to the vector of logits for each training sample.
3. Classify every training sample. In case of a tie, choose any class.
4. Compute the gradient of the function NLL at the point of the initial weight matrix.
5. Update the initial weight matrix applying gradient descent with learning rate $\eta = 1.0$.

Solution:

0. First, we must mind that the training samples are provided by rows. Then, we have to prepare the training samples to be ready to be applied in logistic regression. To this purpose, we convert the training samples into homogeneous notation and class labels into one-hot encoding:

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

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

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

1. Vector of logits for each training sample, that is, $\boldsymbol{a_1}$ 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. Applying the softmax function to obtain $\boldsymbol{\mu}_1$ 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. Classification of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$:

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

4. Computing the gradient of the NLL function at the point of the initial weight matrix:

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

5. Updated the weight matrix:

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