T3.2 Regresión logística

Índice

1 Introducción

Regresión logística: clasificador discriminativo para C clases

Regresión logística binaria: $C=2$

Regresión logística multinomial o multiclase: $C>2$

2 Regresión logística binaria

2.1 Modelo

Regresión logística binaria: Bernoulli condicional para clasificación binaria, $y \in \{0, 1\}$,

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Ber}(y \mid \sigma(a)),$$

de log-odds lineal con la entrada,

$$a = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}^t \boldsymbol{x} + b,$$

por lo que

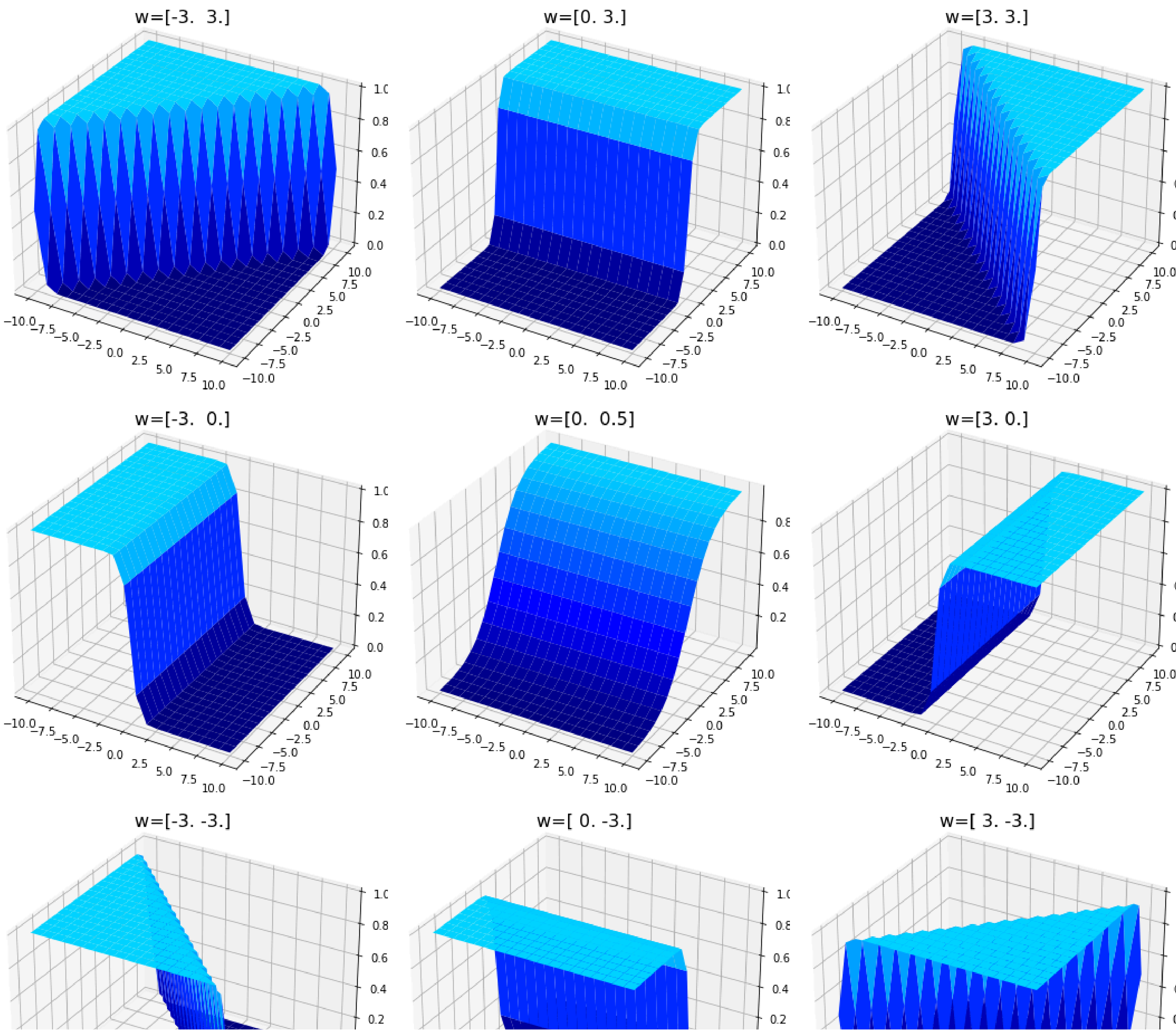
$$\begin{aligned} p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) &= \sigma(a) = \frac{1}{1 + e^{-a}} \\ p(y = 0 \mid \boldsymbol{x}; \boldsymbol{\theta}) &= 1 - \sigma(a) = \sigma(-a) = \frac{1}{1 + e^{a}} \end{aligned}$$

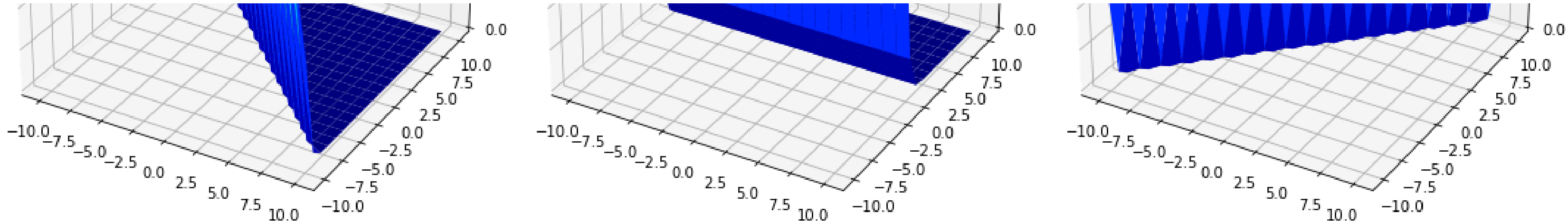
Con etiquetas $\tilde{y} \in \{-1, 1\}$,

$$p(\tilde{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\tilde{y} a)$$

Ejemplo: $p(y = 1 \mid x_1, x_2; \boldsymbol{w}) = \sigma(w_1 x_1 + w_2 x_2)$ para varios $\boldsymbol{w}$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
x, y = np.meshgrid(np.linspace(-10, 10, 20), np.linspace(-10, 10, 20))
w = np.array([ [-3, 3], [0, 3], [3, 3], [-3, 0], [0, 0.5], [3, 0], [-3, -3], [0, -3], [3, -3] ])
nrows = ncols = int(np.ceil(np.sqrt(len(w))))
fig, axes = plt.subplots(nrows, ncols, figsize=(20/4*ncols, 20/4*nrows), constrained_layout=True)
for i in np.arange(len(w)):
    ax = axes.flat[i]; ax.axis('off')
    ax = fig.add_subplot(nrows, ncols, i + 1, projection='3d')
    z = 1.0 / (1.0 + np.exp(-(w[i, 0] * x + w[i, 1] * y)))
    ax.plot_surface(x, y, z, cmap='jet', vmin=0, vmax=3, rstride=1, cstride=1, linewidth=0)
    ax.set_title('w={0!s:.21s}'.format(w[i]), fontsize = 16, y=1)
```


w=[-3. 3.]
w=[0. 3.]
w=[3. 3.]
w=[-3. 0.]
w=[0. 0.5]
w=[3. 0.]
w=[-3. -3.]
w=[0. -3.]
w=[3. -3.]



2.2 Clasificadores lineales

La regla de decisión MAP para regresión logística binaria puede expresarse en función de la logodds como sigue:

$$\begin{aligned}f(\boldsymbol{x})&=\mathbb{I}(p(y=1\mid\boldsymbol{x})>p(y=0\mid\boldsymbol{x}))\\&=\mathbb{I}\left(\log\frac{p(y=1\mid\boldsymbol{x})}{p(y=0\mid\boldsymbol{x})}>0\right)\\&=\mathbb{I}(a>0)\quad\text{con}\quad a=\boldsymbol{w}^t\boldsymbol{x}+b\end{aligned}$$

Por tanto, esta regla viene a ser una función predictora lineal,

$$f(\boldsymbol{x};\boldsymbol{\theta})=b+\boldsymbol{w}^t\boldsymbol{x}=b+\sum_{d=1}^D w_dx_d$$

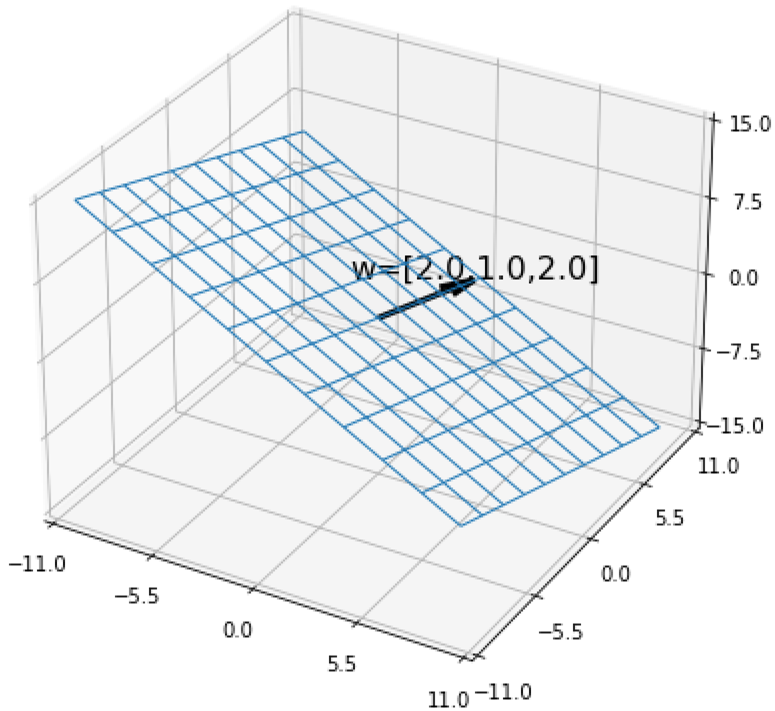
que separa el espacio de entrada en dos partes mediante una frontera hiperplanar,

$$\boldsymbol{w}^t\boldsymbol{x}+b=0$$

Ejemplo: $\boldsymbol{w}=(2,1,2)^t$ y $b=0$; frontera $2x_1+x_2+2x_3+0=0$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
w1, w2, w3, b = 2.0, 1.0, 2.0, 0.0
x1, x2 = np.meshgrid(np.linspace(-10, 10, 11), np.linspace(-10, 10, 11))
x3 = lambda x1, x2: (-w1 * x1  - w2 * x2 - b) / w3
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(x1, x2, x3(x1, x2), rstride=1, cstride=1, linewidth=1)
```

```
scaw = 2.0; ax.quiver(0, 0, x3(0, 0), scaw * w1, scaw * w2, scaw * w3, linewidth=3, colors='black')
ax.text(scaw * w1, scaw * w2, scaw * w3, f"w=[{w1},{w2},{w3}]", fontsize=16, ha='center')
x_min, x_max = ax.get_xlim(); ax.set_xticks(np.linspace(x_min, x_max, 5))
y_min, y_max = ax.get_ylim(); ax.set_yticks(np.linspace(y_min, y_max, 5))
z_min, z_max = ax.get_zlim(); ax.set_zticks(np.linspace(z_min, z_max, 5));
```

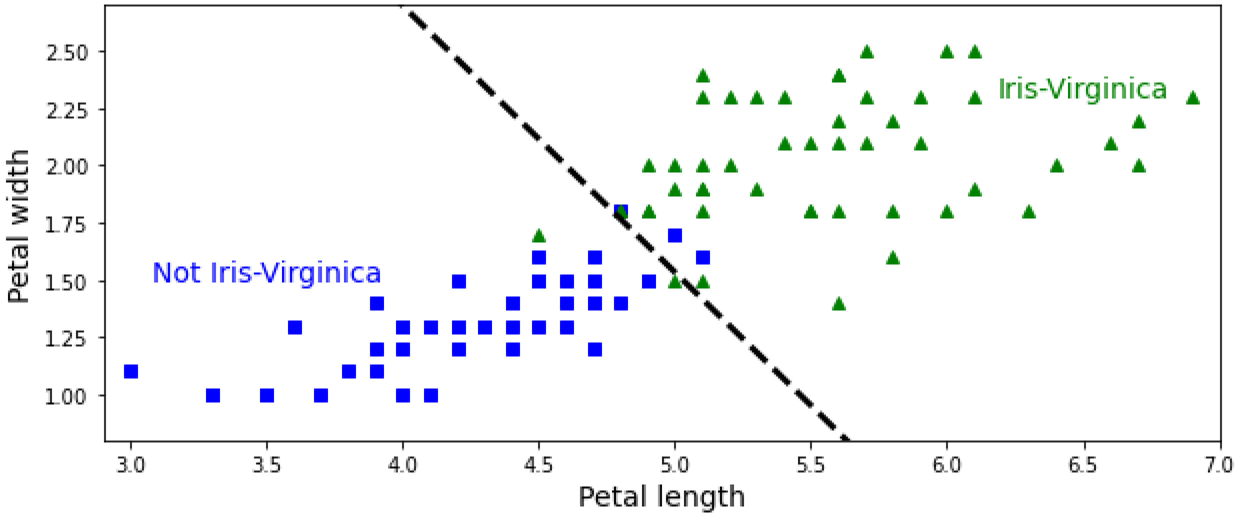


Separabilidad lineal: decimos que las muestras (de entrenamiento) son **linealmente separables** si pueden separarse mediante un hiperplano

Ejemplo: virgínica y no-virgínica no son separables con longitud y amplitud de pétalos

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
iris = load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
```

```
y = np.array(iris["target"] == 2).astype(int)  # 1 if Iris-Virginica, else 0
log_reg = LogisticRegression(solver="lbfgs").fit(X, y)
plt.figure(figsize=(10, 4))
plt.plot(X[y == 0, 0], X[y == 0, 1], "bs")
plt.plot(X[y == 1, 0], X[y == 1, 1], "g^")
left_right = np.array([2.9, 7])
boundary = -(log_reg.coef_[0][0] * left_right + log_reg.intercept_[0]) / log_reg.coef_[0][1]
plt.plot(left_right, boundary, "k--", linewidth=3)
plt.text(3.5, 1.5, "Not Iris-Virginica", fontsize=14, color="b", ha="center")
plt.text(6.5, 2.3, "Iris-Virginica", fontsize=14, color="g", ha="center")
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.axis([2.9, 7, 0.8, 2.7]);
```



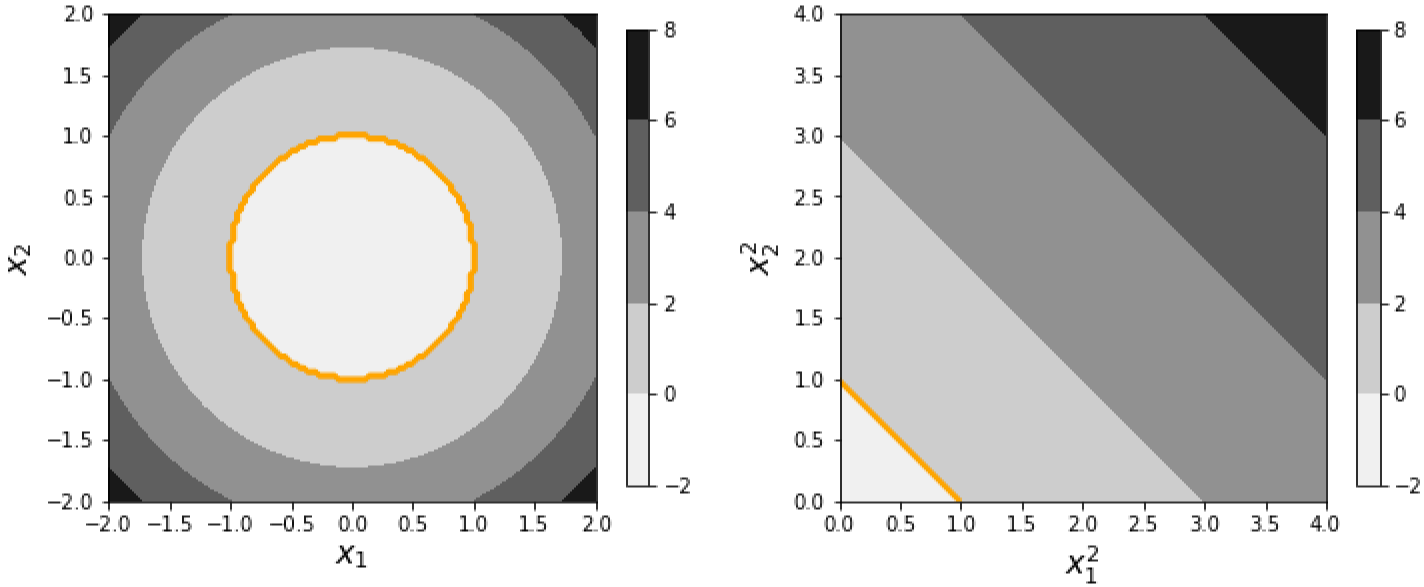
2.3 Clasificadores no lineales

No-linealidad usual: de los problemas de clasificación; esto es, con datos de entrenamiento no linealmente separables

Linearización: estrategia usual para atacar un problema no lineal: **linearizar** los datos en preproceso

Ejemplo: $f(\boldsymbol{x}) = x_1^2 + x_2^2 - R^2 = \boldsymbol{w}^t\phi(\boldsymbol{x}) + b$ con preproceso $\phi(x_1, x_2) = (x_1^2, x_2^2),\ \boldsymbol{w} = (1, 1)$ y $b = -R^2$

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
R = 1
x1, x2 = np.meshgrid(np.linspace(-2, 2, num=128), np.linspace(-2, 2, num=128))
X = np.c_[np.ravel(x1), np.ravel(x2)]
z = lambda x: x[0]**2 + x[1]**2 - R**2
Z = np.apply_along_axis(z, 1, X)
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
axes[0].set(aspect='equal')
axes[0].set_xlabel('$x_1$', fontsize=16); axes[0].set_ylabel('$x_2$', fontsize=16)
axes[0].contour(x1, x2, (Z > 0).reshape(x1.shape), 4, colors='orange', linestyles='solid')
cp = axes[0].contourf(x1, x2, Z.reshape(x1.shape), 4, cmap='Greys')
plt.colorbar(cp, ax=axes[0], shrink=0.7);
xx1, xx2 = np.meshgrid(np.linspace(0, 4, num=128), np.linspace(0, 4, num=128))
XX = np.c_[np.ravel(xx1), np.ravel(xx2)]
zz = lambda xx: xx[0] + xx[1] - R**2
ZZ = np.apply_along_axis(zz, 1, XX)
axes[1].set(aspect='equal')
axes[1].set_xlabel('$x_1^2$', fontsize=16); axes[1].set_ylabel('$x_2^2$', fontsize=16)
axes[1].contour(xx1, xx2, (ZZ > 0).reshape(xx1.shape), 4, colors='orange', linestyles='solid')
cp = axes[1].contourf(xx1, xx2, ZZ.reshape(xx1.shape), 4, cmap='Greys')
plt.colorbar(cp, ax=axes[1], shrink=0.7);
```



2.4 Estimación máximo-verosímil

Sea un modelo de regresión logística binaria $p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Ber}(y \mid \mu)$, $y \in \{0, 1\}$, con $\mu = \sigma(a)$ y $a = \boldsymbol{w}^t \boldsymbol{x}$, en el que asumimos que $\boldsymbol{w}$ absorbe el sesgo b. La neg-log-verosimilitud de $\boldsymbol{w}$ respecto a N datos $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}$ (normalizada por N) es:

$$\begin{aligned}
\operatorname{NLL}(\boldsymbol{w}) &= -\frac{1}{N} \log p(\mathcal{D} \mid \boldsymbol{w}) \\
&= -\frac{1}{N} \log \prod_{n=1}^{N} \operatorname{Ber}(y_n \mid \mu_n) && (\mu_n = \sigma(a_n) \text{ con log-odds } a_n = \boldsymbol{w}^t \boldsymbol{x}_n) \\
&= -\frac{1}{N} \sum_{n=1}^{N} \log(\mu_n^{y_n} (1 - \mu_n)^{(1-y_n)}) \\
&= -\frac{1}{N} \sum_{n=1}^{N} y_n \log \mu_n + (1 - y_n) \log(1 - \mu_n) \\
&= \frac{1}{N} \sum_{n=1}^{N} \mathbb{H}(y_n, \mu_n) && (\mathbb{H} \text{ entropía cruzada})
\end{aligned}$$

Es fácil comprobar que el gradiente del objetivo es:

$$\nabla_{\boldsymbol{w}} \operatorname{NLL}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} (\mu_n - y_n) \boldsymbol{x}_n$$

Una manera sencilla de minimizar el objetivo consiste en aplicar descenso por gradiente estocástico con minibatch de talla uno:

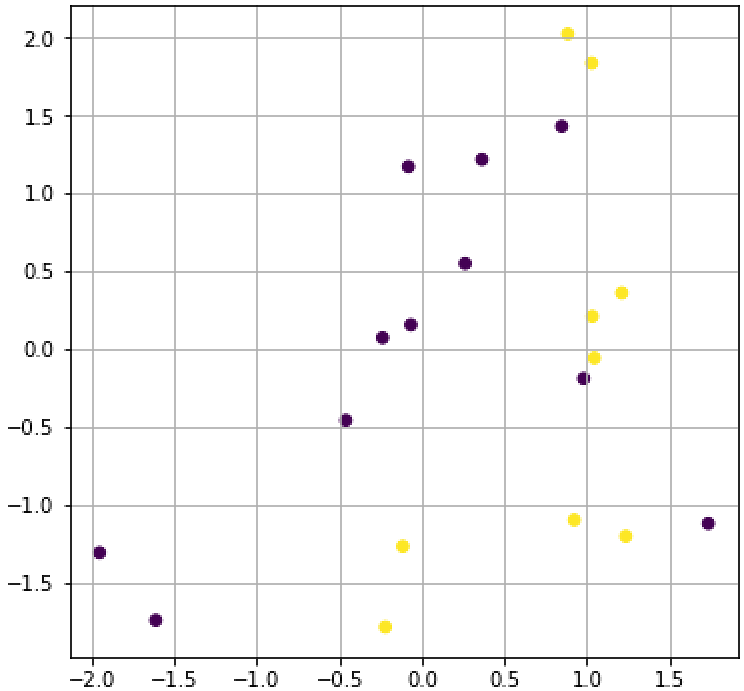
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t (\mu_n - y_n) \boldsymbol{x}_n$$

Ejemplo: datos sintéticos 2d y modelo de sesgo nulo ($b = 0$)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
N, n_clusters_per_class, class_sep = 20, 2, 1.0
X, y = make_classification(n_samples=N, n_features=2, n_redundant=0, n_classes=2,
    n_clusters_per_class=n_clusters_per_class, class_sep=class_sep) #, random_state=1)
print(np.c_[X, y])
```

```
[[ 1.735221   -1.12112183  0.        ]
 [-0.11594029 -1.26769196  1.        ]
 [ 0.88381226  2.01952423  1.        ]
 [ 0.92291618 -1.09847542  1.        ]
 [ 0.98006072 -0.19121314  0.        ]
 [-1.95367795 -1.30843893  0.        ]
 [-0.23871912  0.07069383  0.        ]
 [ 0.36346196  1.21438005  0.        ]
 [-0.46122945 -0.45937469  0.        ]
 [-0.06719974  0.15370342  0.        ]
 [-0.22177128 -1.78498188  1.        ]
 [ 1.23556323 -1.20291205  1.        ]
 [ 0.84719434  1.42690919  0.        ]
 [ 1.0448741  -0.0593187   1.        ]
 [-1.61288152 -1.74205102  0.        ]
 [ 1.02868227  1.83257538  1.        ]
 [ 0.26196032  0.54697936  0.        ]
 [ 1.21097653  0.35805219  1.        ]
 [-0.08226719  1.16827966  0.        ]
 [ 1.03206805  0.20762139  1.        ]]
```

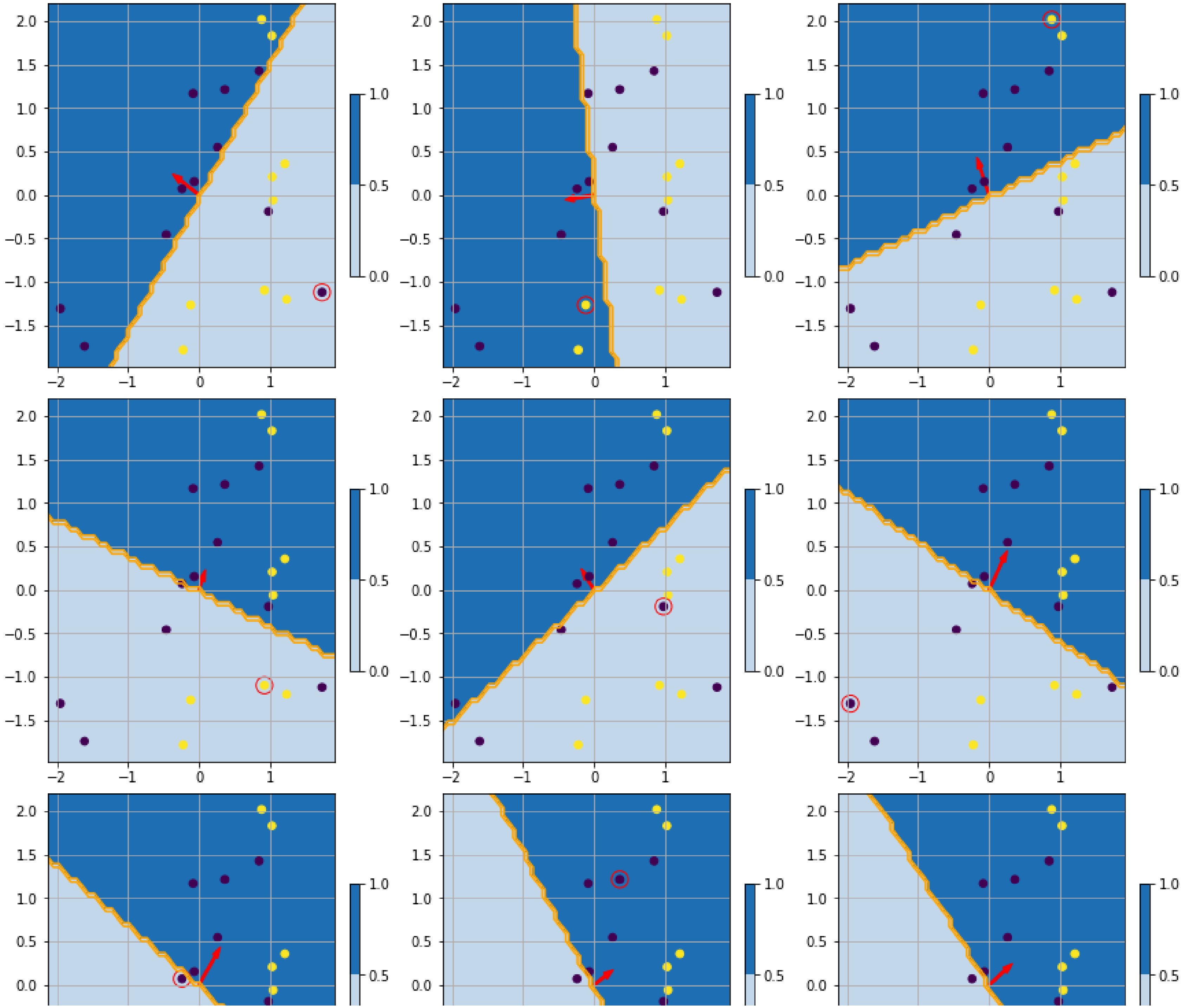
In [2]:
```
fig, ax = plt.subplots(figsize=(6, 6)); ax.grid(); ax.scatter(*X.T, c=y, s=32)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50), np.linspace(y_min, y_max, 50))
XX = np.c_[np.ravel(xx), np.ravel(yy)]
```

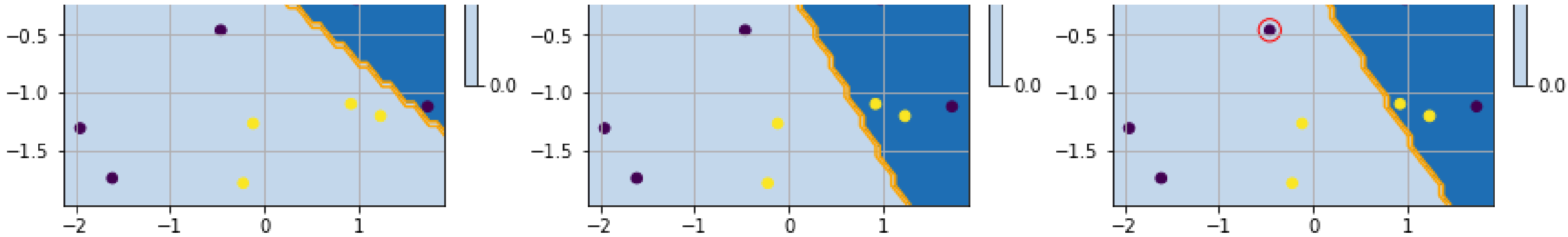


In [3]:
```
w, eta = np.zeros((N + 1, 2)), 0.3
for n in np.arange(N):
    mun = 1.0 / (1.0 + np.exp(- w[n, :] @ X[n, :]))
    grad = mun - y[n]
    w[n+1, :] = w[n, :] - eta * (mun - y[n]) * X[n, :]
    print(n+1, w[n+1])
```

```
1 [-0.26028315  0.16816827]
2 [-0.27926112 -0.03933698]
3 [-0.12525289  0.31257414]
4 [0.04440654 0.1106417 ]
5 [-0.10424642  0.1396444 ]
6 [0.19187455 0.33796583]
7 [0.22729012 0.32747793]
8 [0.1599242  0.10239843]
9 [0.22493491 0.16714772]
10 [0.23506817 0.14397029]
11 [ 0.19670156 -0.16483304]
12 [ 0.34179149 -0.3060888 ]
13 [ 0.22404833 -0.50440079]
14 [ 0.36020852 -0.51213076]
15 [ 0.63948264 -0.21049064]
16 [0.7729219  0.02722852]
17 [ 0.72937394 -0.06370063]
18 [ 0.83736134 -0.03177175]
19 [ 0.84904797 -0.19773407]
20 [ 0.9427212  -0.17888981]
```

In [4]:
```python
nrows = ncols = int(min(3, np.ceil(np.sqrt(N))));
fig, axes = plt.subplots(nrows, ncols, figsize=(12, 12), constrained_layout=True)
for n in np.arange(min(N, nrows * ncols)):
    ax = axes.flat[n]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, n + 1); ax.grid()
    z = lambda x: w[n + 1, :] @ x
    zz = np.heaviside(np.apply_along_axis(z, 1, XX), 0.0)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 1, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
    ax.arrow(0, 0, w[n+1, 0], w[n+1, 1], width=.03, facecolor='red', edgecolor='red')
    ax.scatter(*X.T, c=y, s=32); ax.scatter(X[n, 0], X[n, 1], facecolors='none', edgecolors='red', s=150)
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
1.0
0.5
0.0



2.5 Perceptrón

Regresión logística binaria es un modelo probabilístico para clasificación en dos clases, $y \in \{0,1\}$,

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Ber}(y \mid \mu) \quad \text{con} \quad \mu = \sigma(a) \quad \text{y} \quad a = \boldsymbol{w}^t\boldsymbol{x} \quad (b \text{ absorbido en } \boldsymbol{w})$$

Perceptrón puede verse como una variante con escalón Heaviside, $H(a) = \mathbb{I}(a > 0)$, en lugar de sigmoide:

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Ber}(y \mid \mu) \quad \text{con} \quad \mu = H(a) \quad \text{y} \quad a = \boldsymbol{w}^t\boldsymbol{x} \quad (b \text{ absorbido en } \boldsymbol{w})$$

En el caso de regresión logística, el MLE de $\boldsymbol{w}$ puede obtenerse mediante descenso por gradiente estocástico (con minibatch de talla uno):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t(\mu_n - y_n)\boldsymbol{x}_n$$

En el caso de Perceptrón, el MLE de $\boldsymbol{w}$ no puede obtenerse del mismo modo ya que la log-verosimilitud no es diferenciable. No obstante, $\boldsymbol{w}$ puede aprenderse mediante el **algoritmo Perceptrón**, iterando sobre los datos con:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t(\hat{y}_n - y_n)\boldsymbol{x}_n$$

Nótese que el algoritmo Perceptrón es prácticamente idéntico a SGD aplicado a regresión logística binaria.

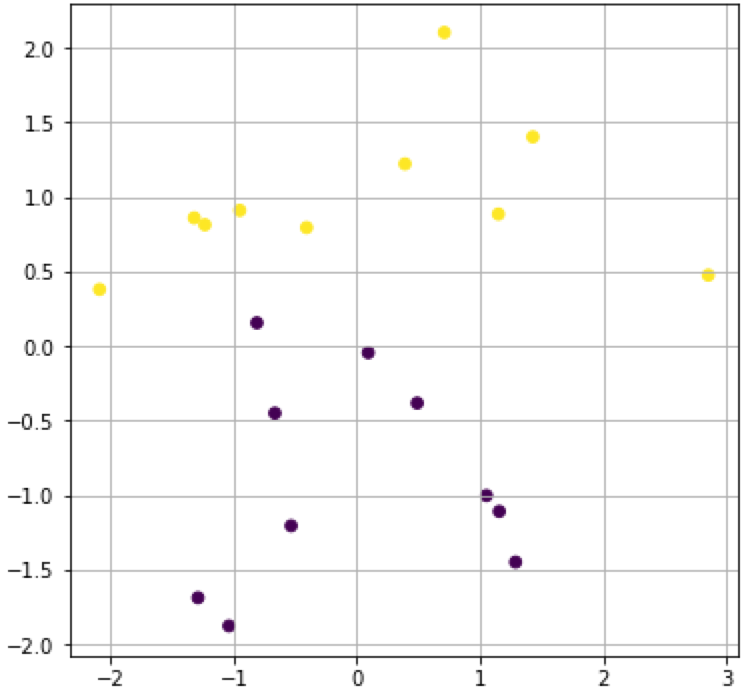
Ejemplo: datos sintéticos 2d y modelo de sesgo nulo ($b = 0$)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
```

```
N, n_clusters_per_class, class_sep = 20, 2, 1.0
X, y = make_classification(n_samples=N, n_features=2, n_redundant=0, n_classes=2,
    n_clusters_per_class=n_clusters_per_class, class_sep=class_sep) #, random_state=1)
print(np.c_[X, y])
```

```
[[ 1.42921352  1.40216373  1.        ]
 [-0.53343274 -1.20720614  0.        ]
 [-1.28955298 -1.68987396  0.        ]
 [-0.81129651  0.15449015  0.        ]
 [-2.08864582  0.37869555  1.        ]
 [ 1.28899648 -1.45092469  0.        ]
 [-1.23343394  0.81249019  1.        ]
 [ 0.71065664  2.10285935  1.        ]
 [ 1.05265208 -1.00439831  0.        ]
 [ 1.15656228 -1.11089165  0.        ]
 [-1.03817785 -1.87913803  0.        ]
 [-0.95023001  0.90895826  1.        ]
 [ 2.8537504   0.47453234  1.        ]
 [-1.32010482  0.85871803  1.        ]
 [ 0.09268322 -0.04590865  0.        ]
 [ 0.39125814  1.22104917  1.        ]
 [ 0.49027352 -0.38395735  0.        ]
 [ 1.1495338   0.88425079  1.        ]
 [-0.6651082  -0.45171466  0.        ]
 [-0.40759602  0.79371629  1.        ]]
```

In [2]:
```
fig, ax = plt.subplots(figsize=(6, 6)); ax.grid(); ax.scatter(*X.T, c=y, s=32)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50), np.linspace(y_min, y_max, 50))
XX = np.c_[np.ravel(xx), np.ravel(yy)]
```

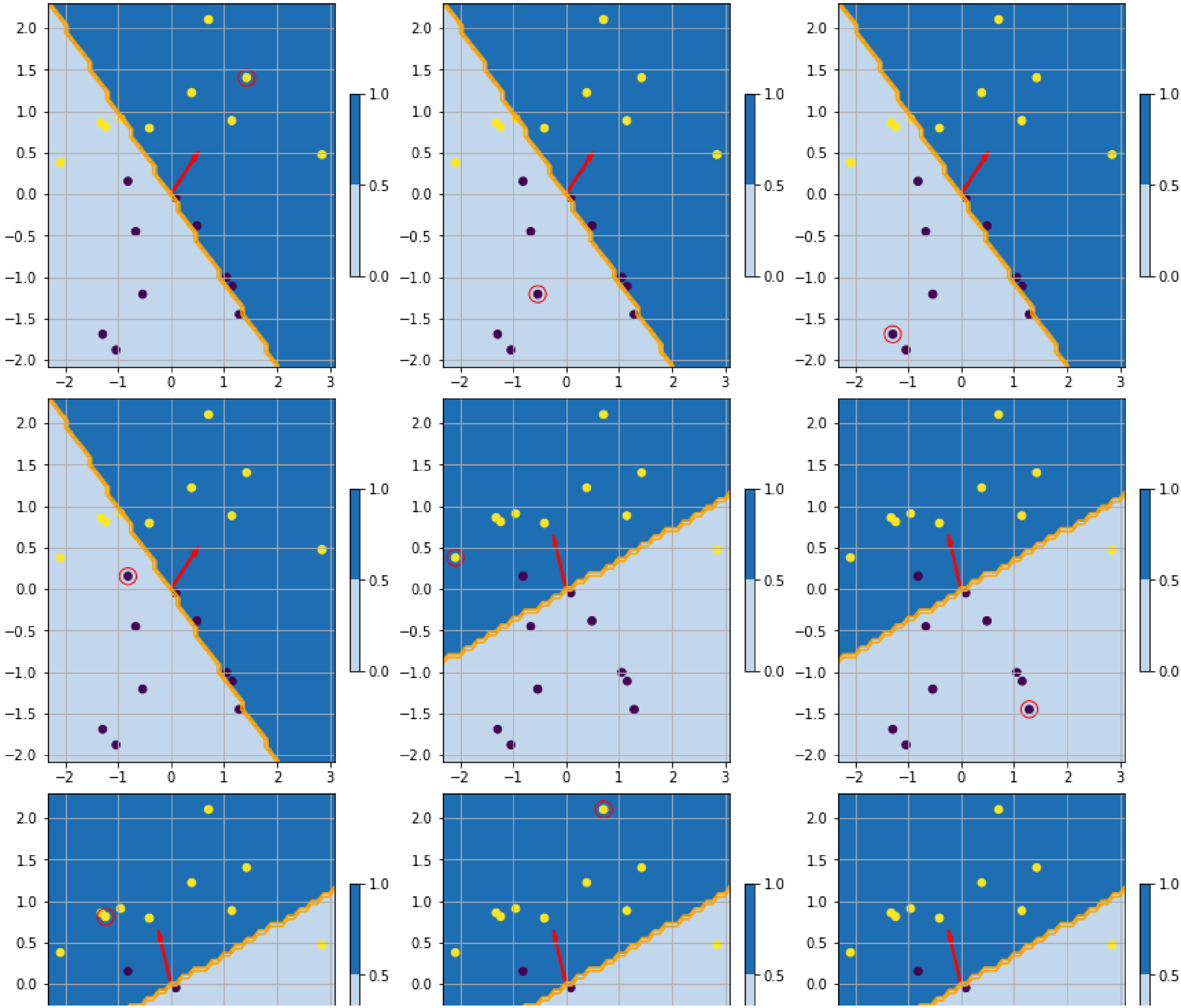


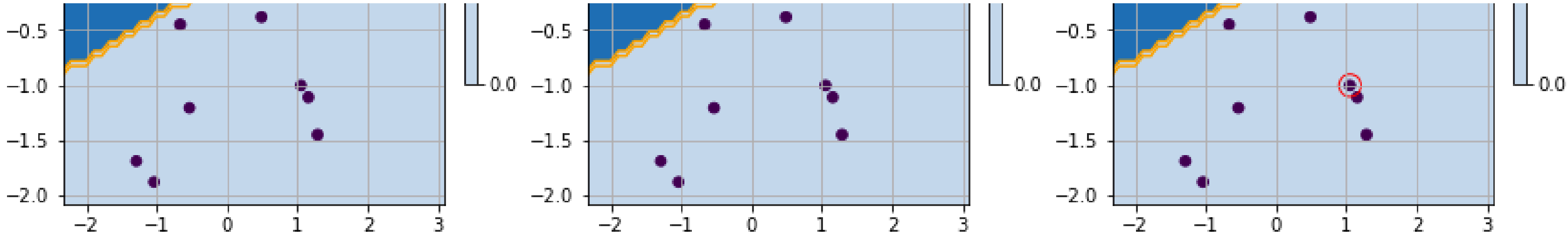
In [3]:
```
w, eta = np.zeros((N + 1, 2)), 0.3
for n in np.arange(N):
    # mun = 1.0 / (1.0 + np.exp(- w[n, :] @ X[n, :]))
    mun = np.heaviside(w[n, :] @ X[n, :], 0.0)
    grad = mun - y[n]
    w[n+1, :] = w[n, :] - eta * (mun - y[n]) * X[n, :]
    print(n+1, mun, w[n+1])
```

```
1 0.0 [0.42876406 0.42064912]
2 0.0 [0.42876406 0.42064912]
3 0.0 [0.42876406 0.42064912]
4 0.0 [0.42876406 0.42064912]
5 0.0 [-0.19782969  0.53425778]
6 0.0 [-0.19782969  0.53425778]
7 1.0 [-0.19782969  0.53425778]
8 1.0 [-0.19782969  0.53425778]
9 0.0 [-0.19782969  0.53425778]
10 0.0 [-0.19782969  0.53425778]
11 0.0 [-0.19782969  0.53425778]
12 1.0 [-0.19782969  0.53425778]
13 0.0 [0.65829543 0.67661749]
14 0.0 [0.26226398 0.9342329 ]
15 0.0 [0.26226398 0.9342329 ]
16 1.0 [0.26226398 0.9342329 ]
17 0.0 [0.26226398 0.9342329 ]
18 1.0 [0.26226398 0.9342329 ]
19 0.0 [0.26226398 0.9342329 ]
20 1.0 [0.26226398 0.9342329 ]
```

In [4]:

```
nrows = ncols = int(min(3, np.ceil(np.sqrt(N))));
fig, axes = plt.subplots(nrows, ncols, figsize=(12, 12), constrained_layout=True)
for n in np.arange(min(N, nrows * ncols)):
    ax = axes.flat[n]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, n + 1); ax.grid()
    z = lambda x: w[n + 1, :] @ x
    zz = np.heaviside(np.apply_along_axis(z, 1, XX), 0.0)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 1, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
    ax.arrow(0, 0, w[n+1, 0], w[n+1, 1], width=.03, facecolor='red', edgecolor='red')
    ax.scatter(*X.T, c=y, s=32); ax.scatter(X[n, 0], X[n, 1], facecolors='none', edgecolors='red', s=150)
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3
1.0
0.5
0.0



2.6 Estimación MAP

La regularización ℓ_2 de regresión logística binaria consiste en asumir un prior Gaussiano para $\boldsymbol{w}$,

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \mathbf{0}, \lambda^{-1}\mathbf{I})$$

y minimizar la log-verosimilitud negativa penalizada para hallar un estimador MAP de $\boldsymbol{w}$,

$$\begin{aligned}
\boldsymbol{w}_{\text{map}} &= \operatorname*{argmax}_{\boldsymbol{w}}\; p(\boldsymbol{w} \mid \mathcal{D}) \\
&= \operatorname*{argmax}_{\boldsymbol{w}}\; \log p(\mathcal{D} \mid \boldsymbol{w}) + \log p(\boldsymbol{w}) \\
&= \operatorname*{argmax}_{\boldsymbol{w}}\; \operatorname{LL}(\boldsymbol{w}) - \lambda \boldsymbol{w}^t \boldsymbol{w} \\
&= \operatorname*{argmin}_{\boldsymbol{w}}\; \operatorname{PNLL}(\boldsymbol{w}) \qquad \text{con} \qquad \operatorname{PNLL}(\boldsymbol{w}) = \operatorname{NLL}(\boldsymbol{w}) + \lambda \boldsymbol{w}^t \boldsymbol{w}
\end{aligned}$$

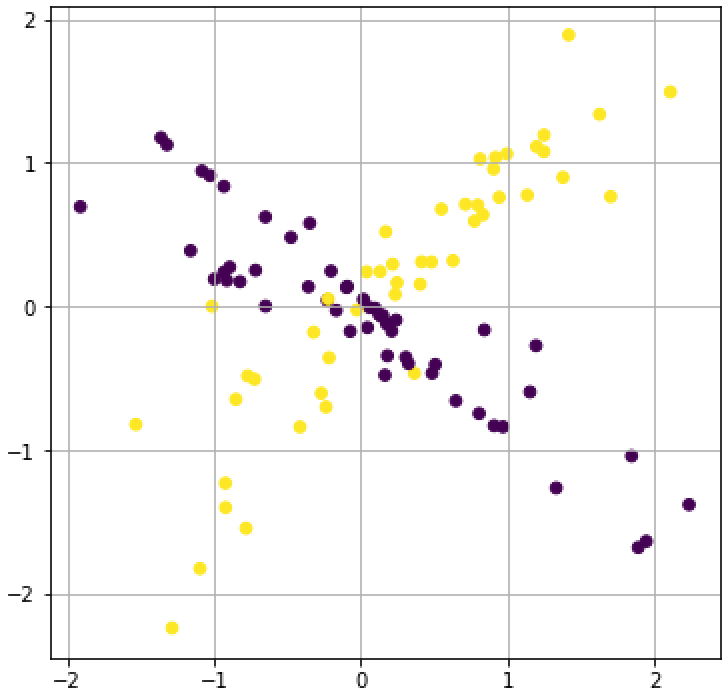
Ejemplo: datos sintéticos 2d y modelo polinómico

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
Ntrain, Ntest, n_clusters_per_class, class_sep = 100, 1000, 2, 0.1 # Ntrain = 50 en PML1
N = Ntrain + Ntest
X, y = make_classification(n_samples=N, n_features=2, n_redundant=0, n_classes=2,
    n_clusters_per_class=n_clusters_per_class, class_sep=class_sep, random_state=1)
Xtrain = X[:Ntrain, :]; ytrain = y[:Ntrain]
```

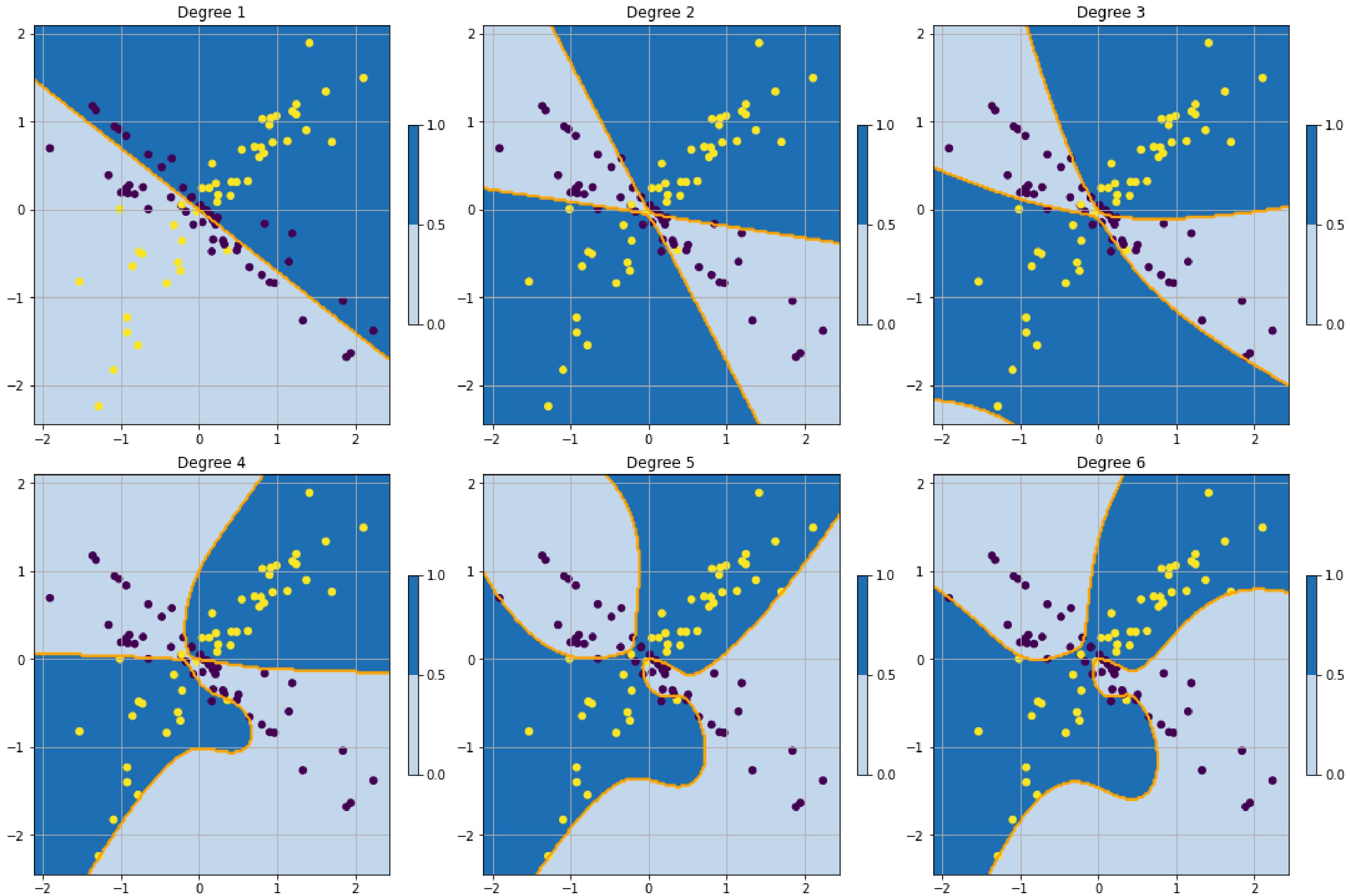
```
Xtest = X[Ntrain:, :]; ytest = y[Ntrain:]
print(np.c_[Xtrain[:min(Ntrain, 10), :], ytrain[:min(Ntrain, 10)]])
```

```
[[-6.48928917e-01  1.96408596e-03  0.00000000e+00]
 [-2.69137092e-01 -6.04600245e-01  1.00000000e+00]
 [ 8.06035976e-01 -7.45521952e-01  0.00000000e+00]
 [-8.22767535e-01  1.71948873e-01  0.00000000e+00]
 [-4.75851630e-01  4.79774731e-01  0.00000000e+00]
 [ 9.17638363e-01  1.03699227e+00  1.00000000e+00]
 [ 4.80934984e-01  3.09073076e-01  1.00000000e+00]
 [ 7.12475381e-01  7.10609599e-01  1.00000000e+00]
 [-1.01316079e+00  8.37223758e-04  1.00000000e+00]
 [ 1.70220224e+00  7.64383740e-01  1.00000000e+00]]
```

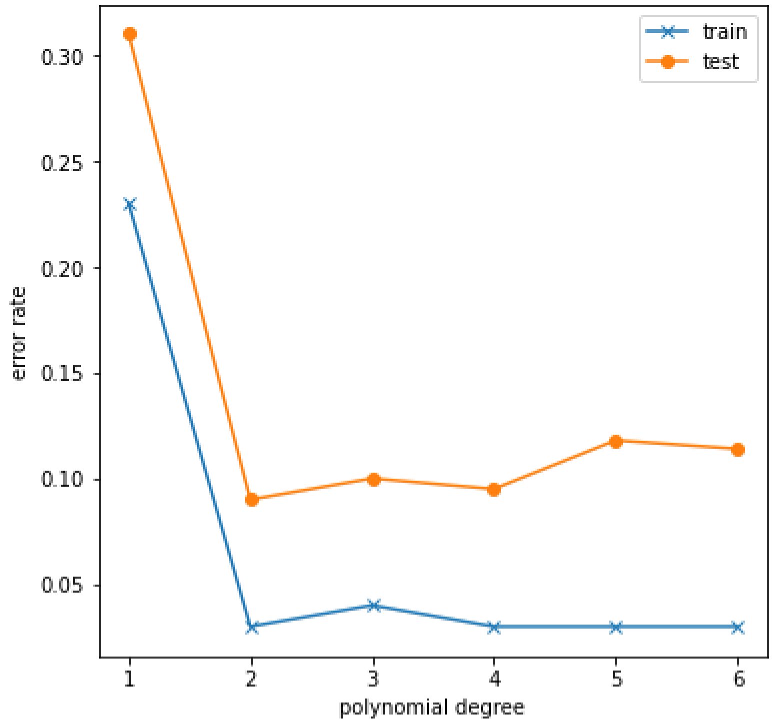
In [2]:
```
fig, ax = plt.subplots(figsize=(6, 6)); ax.grid(); ax.scatter(*Xtrain.T, c=ytrain, s=32)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
XX = np.c_[np.ravel(xx), np.ravel(yy)]
```



In [3]:
```
degrees = [1, 2, 3, 4, 5, 6]; nrows, ncols = 2, 3
acc_train = np.zeros(len(degrees)); acc_test = np.zeros(len(degrees))
C=1e4 # C = 1 / lambda: varianza del prior
fig, axes = plt.subplots(nrows, ncols, figsize=(15, 10), constrained_layout=True)
for i, degree in enumerate(degrees):
    ax = axes.flat[i]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, i + 1); ax.grid()
    transformer = PolynomialFeatures(degree)
    Xtrain_poly = transformer.fit_transform(Xtrain)[:, 1:] # skip the first column of 1s
    model = LogisticRegression(C=C, max_iter=1000)
    model = model.fit(Xtrain_poly, ytrain)
    acc_train[i] = accuracy_score(ytrain, model.predict(Xtrain_poly))
    Xtest_poly = transformer.fit_transform(Xtest)[:, 1:] # skip the first column of 1s
    acc_test[i] = accuracy_score(ytest, model.predict(Xtest_poly))
    XX_poly = transformer.fit_transform(XX)[:, 1:] # skip the first column of 1s
    z = lambda x: model.coef_[0] @ x
    zz = np.heaviside(np.apply_along_axis(z, 1, XX_poly), 0.0)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 1, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
    ax.scatter(*Xtrain.T, c=ytrain, s=32)
    ax.set_title('Degree {}'.format(degree))
```

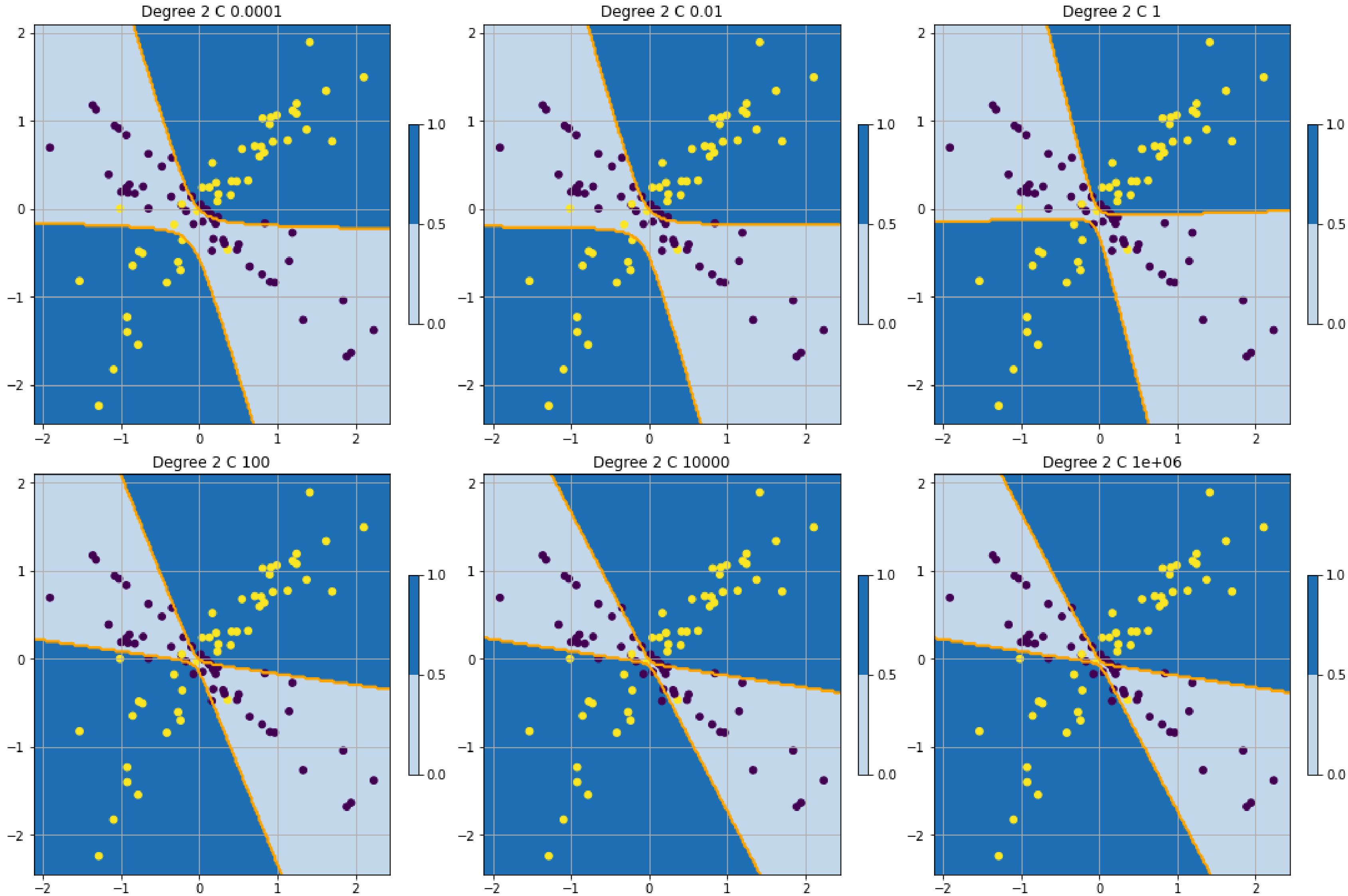


In [4]:
```
plt.figure(figsize=(6, 6))
plt.plot(degrees, 1.0 - acc_train, 'x-', label='train')
plt.plot(degrees, 1.0 - acc_test, 'o-', label='test')
plt.legend()
plt.xlabel('polynomial degree')
plt.ylabel('error rate');
```



In [5]:
```
degree = 2; Cs = [1e-4, 1e-2, 1e0, 1e2, 1e4, 1e6]; nrows, ncols = 2, 3
acc_train = np.zeros(len(Cs)); acc_test = np.zeros(len(Cs))
fig, axes = plt.subplots(nrows, ncols, figsize=(15, 10), constrained_layout=True)
for i, C in enumerate(Cs):
    ax = axes.flat[i]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, i + 1); ax.grid()
    transformer = PolynomialFeatures(degree)
    Xtrain_poly = transformer.fit_transform(Xtrain)[:, 1:] # skip the first column of 1s
    model = LogisticRegression(C=Cs[i], max_iter=1000)
    model = model.fit(Xtrain_poly, ytrain)
    acc_train[i] = accuracy_score(ytrain, model.predict(Xtrain_poly))
    Xtest_poly = transformer.fit_transform(Xtest)[:, 1:] # skip the first column of 1s
    acc_test[i] = accuracy_score(ytest, model.predict(Xtest_poly))
    XX_poly = transformer.fit_transform(XX)[:, 1:] # skip the first column of 1s
    z = lambda x: model.coef_[0] @ x
    zz = np.heaviside(np.apply_along_axis(z, 1, XX_poly), 0.0)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 1, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
```

```
    ax.scatter(*Xtrain.T, c=ytrain, s=32)
    ax.set_title('Degree {} C {:g}'.format(degree, Cs[i]))
```



3 Regresión logística multiclase

3.1 Modelo

Regresión logística multinomial es una categórica condicional para clasificación multiclase, $y \in \{1, \dots, C\}$,

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Cat}(y \mid S(\boldsymbol{a})),$$

de logits lineales con la entrada,

$$\boldsymbol{a} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \mathbf{W}^t \boldsymbol{x} + \boldsymbol{b} \qquad \text{con} \qquad \boldsymbol{\theta} = (\mathbf{W}, \boldsymbol{b}), \quad \mathbf{W} \in \mathbb{R}^{D \times C}, \quad \boldsymbol{b} \in \mathbb{R}^D$$

En notación homogénea, anteponiendo un 1 a $\boldsymbol{x}$ y $\boldsymbol{b}$ a $\mathbf{W}$,

$$\boldsymbol{a} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \mathbf{W}^t \boldsymbol{x}$$

Clasificación multiclase vs multi-etiqueta:

- **Clasificación multiclase:** caso estándar en el que solo una etiqueta es correcta
- **Clasificación multi-etiqueta:** se admite que haya cero, una o más etiquetas correctas; suele modelizarse como una extensión de regresión logística binaria donde la salida es un vector de C bits, $\boldsymbol{y} \in \{0, 1\}^C$, para indicar la presencia o ausencia de cada etiqueta

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \prod_{c=1}^{C} \operatorname{Ber}(y_c \mid \sigma(\boldsymbol{w}_c^t \boldsymbol{x})),$$

3.2 Clasificadores lineales y no lineales

Al igual que regresión logística binaria, regresión logística multinomial halla fronteras lineales que, no obstante, pueden emplearse con datos no linealmente separables mediante linearización de los mismos en preproceso.

Ejemplo: $C = 3,\ \boldsymbol{x} = (x_1, x_2)^t,\ \phi(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)^t$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import PolynomialFeatures
from scipy.stats import multivariate_normal as mvn
from sklearn.linear_model import LogisticRegression
import matplotlib.colors as mcol
```

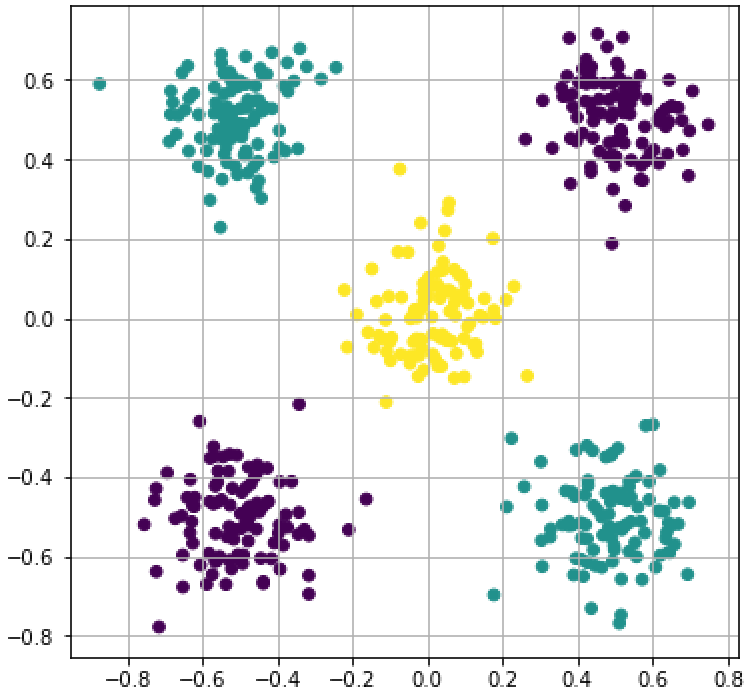
In [2]:

```
np.random.seed(234) # np.random.RandomState(0)
N, S = 100, 0.01 * np.eye(2)
Gs = [mvn(mean=[0.5, 0.5], cov=S), mvn(mean=[-0.5, -0.5], cov=S), mvn(mean=[0.5, -0.5], cov=S), mvn(mean=[
X = np.concatenate([G.rvs(size=N) for G in Gs])
y = np.concatenate((1 * np.ones(N), 1 * np.ones(N), 2 * np.ones(N), 2 * np.ones(N), 3 * np.ones(N)))
print(np.c_[X[:min(N, 10), :], y[:min(N, 10)]])
```

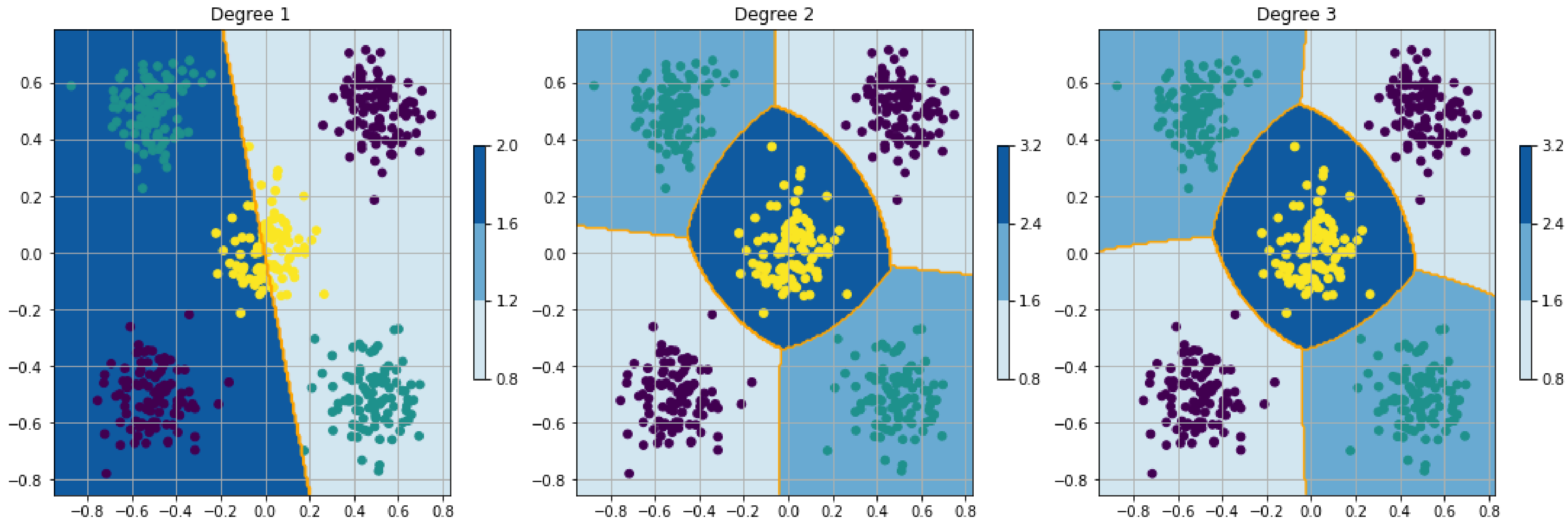
```
[[0.58187916 0.39564494 1.        ]
 [0.53509007 0.59215783 1.        ]
 [0.49126181 0.18711154 1.        ]
 [0.40302673 0.59346658 1.        ]
 [0.50438663 0.64252155 1.        ]
 [0.44429373 0.59268244 1.        ]
 [0.37164463 0.60962569 1.        ]
 [0.30675275 0.54789592 1.        ]
 [0.63445896 0.48245793 1.        ]
 [0.49172956 0.41115453 1.        ]]
```

In [3]:

```
fig, ax = plt.subplots(figsize=(6, 6)); ax.grid(); ax.scatter(*X.T, c=y, s=32)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
XX = np.c_[np.ravel(xx), np.ravel(yy)]
```



In [4]:
```
degrees = [1, 2, 3]; nrows, ncols = 1, 3
C=1e4 # C = 1 / lambda: varianza del prior
fig, axes = plt.subplots(nrows, ncols, figsize=(15, 5), constrained_layout=True)
for i, degree in enumerate(degrees):
    ax = axes.flat[i]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, i + 1); ax.grid()
    transformer = PolynomialFeatures(degree)
    X_poly = transformer.fit_transform(X)[:, 1:] # skip the first column of 1s
    model = LogisticRegression(C=C, max_iter=1000).fit(X_poly, y)
    XX_poly = transformer.fit_transform(XX)[:, 1:] # skip the first column of 1s
    zz = model.predict(XX_poly)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 2, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
    ax.scatter(*X.T, c=y, s=32)
    ax.set_title(f'Degree {degree}')
```



3.3 Estimación máximo-verosímil

Sea un modelo de regresión logística multinomial para C clases, $y \in \{1, \dots, C\}$,

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \operatorname{Cat}(y \mid \boldsymbol{\mu}) \qquad \text{con} \qquad \boldsymbol{\mu} = S(\boldsymbol{a}) \quad \text{y} \quad \boldsymbol{a} = \mathbf{W}^t \boldsymbol{x}$$

donde asumimos que $\mathbf{W} \in \mathbb{R}^{D \times C}$ absorbe el sesgo $\boldsymbol{b}$. La neg-log-verosimilitud de $\mathbf{W}$ respecto a un conjunto de N datos $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ (normalizada por N y con etiquetas one-hot) es:

$$\begin{aligned}
\operatorname{NLL}(\mathbf{W}) &= -\frac{1}{N}\log p(\mathcal{D}\mid\mathbf{W})\\
&= -\frac{1}{N}\log\prod_{n=1}^{N}\operatorname{Cat}(\boldsymbol{y}_n\mid\boldsymbol{\mu}_n) \qquad (\boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n)\text{ con logits }\boldsymbol{a}_n=\mathbf{W}^t\boldsymbol{x}_n)\\
&= -\frac{1}{N}\sum_{n=1}^{N}\log\prod_{c=1}^{C}\mu_{nc}^{y_{nc}}\\
&= -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}y_{nc}\log\mu_{nc}\\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{H}(\boldsymbol{y}_n,\boldsymbol{\mu}_n) \qquad (\mathbb{H}\text{ entropía cruzada})
\end{aligned}$$

Se puede comprobar que el gradiente del objetivo es:

$$\nabla_{\operatorname{vec}(\mathbf{W})}\operatorname{NLL}(\mathbf{W}) = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n(\boldsymbol{\mu}_n-\boldsymbol{y}_n)^t$$

Por comodidad, el gradiente recibe el formato de $\mathbf{W}$, esto es, $\mathbb{R}^{D\times C}$. Si el modelo se define con $\mathbf{W}$ transpuesta, $\mathbf{W}\in\mathbb{R}^{C\times D}$, el formato del gradiente también se transpone.

Una manera sencilla de minimizar el objetivo consiste en aplicar descenso por gradiente estocástico con minibatch de talla uno:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta_t\,\boldsymbol{x}_n(\boldsymbol{\mu}_n-\boldsymbol{y}_n)^t$$

Ejemplo: datos sintéticos 2d y modelo de sesgo nulo ($b=0$)

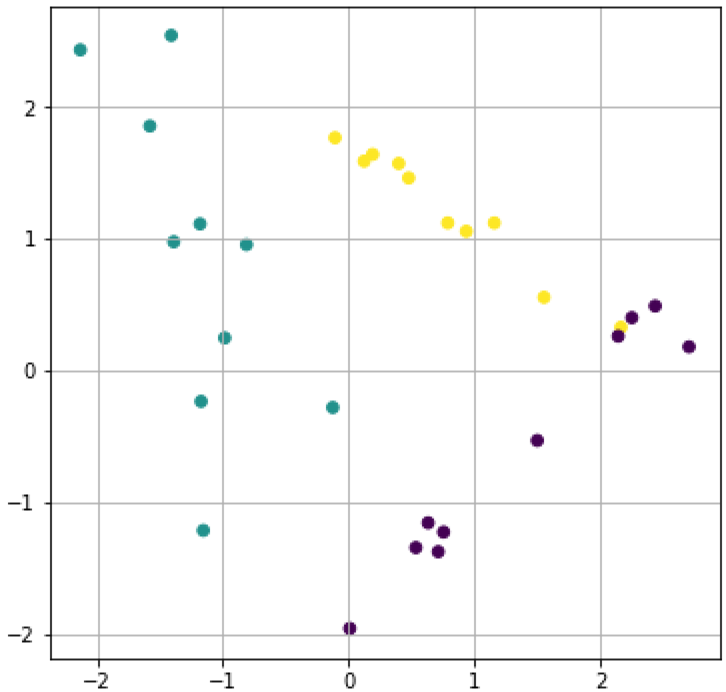
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from scipy.special import logsumexp
N, n_clusters_per_class, class_sep = 30, 1, 1.0
X, y = make_classification(n_samples=N, n_features=2, n_redundant=0, n_classes=3,
```

```
    n_clusters_per_class=n_clusters_per_class, class_sep=class_sep, random_state=43)
print(np.c_[X[:min(N, 10), :], y[:min(N, 10)]])
```

```
[[ 2.1636225   0.32544657  2.        ]
 [-0.98598415  0.24480874  1.        ]
 [ 2.43571903  0.48704515  0.        ]
 [ 0.9342787   1.0535726   2.        ]
 [ 2.25037364  0.3984734   0.        ]
 [-1.39089919  0.97415801  1.        ]
 [ 0.3971162   1.5679538   2.        ]
 [ 0.6309278  -1.15749872  0.        ]
 [ 0.78697279  1.11748862  2.        ]
 [-1.58095132  1.8521708   1.        ]]
```

In [2]:

```
fig, ax = plt.subplots(figsize=(6, 6)); ax.grid(); ax.scatter(*X.T, c=y, s=32)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_max, 200))
XX = np.c_[np.ravel(xx), np.ravel(yy)]
```



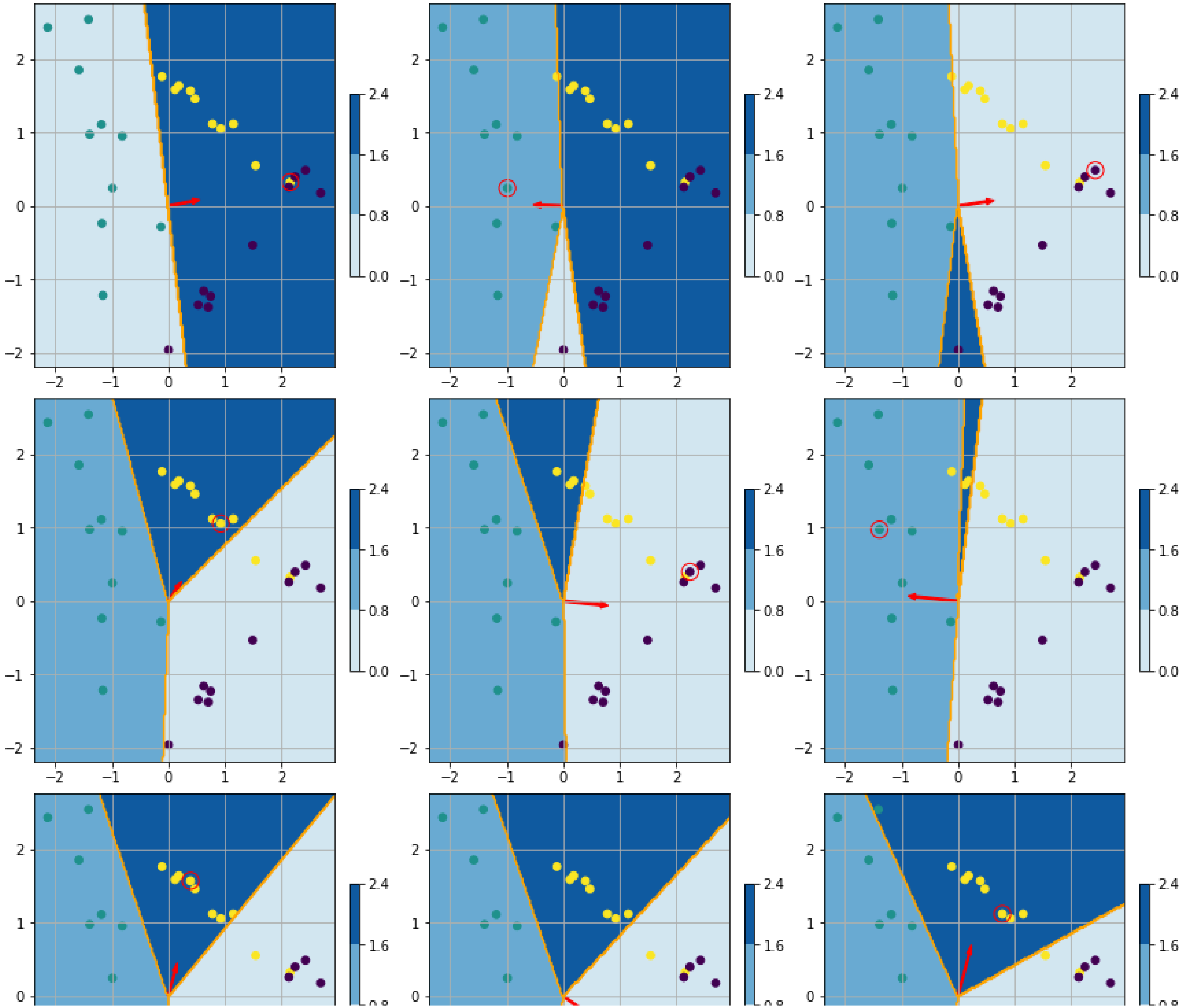
In [3]:

```
W, eta = np.zeros((N + 1, 3, 2)), 0.3
for n in np.arange(N):
    an = W[n, :, :] @ X[n, :]
    mun = np.exp(an - logsumexp(an))
    mun[y[n]] -= 1.0
    W[n+1, :, :] = W[n, :, :] - eta * np.outer(mun, X[n, :])
    if n < 3: print(n+1, W[n+1, :, :])
```

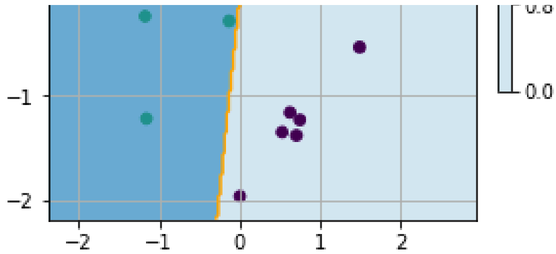
```
1 [[-0.21636225 -0.03254466]
 [-0.21636225 -0.03254466]
 [ 0.4327245   0.06508931]]
2 [[-0.09990987 -0.06145847]
 [-0.39570511  0.01198415]
 [ 0.49561498  0.04947432]]
3 [[ 5.09149935e-01  6.03288219e-02]
 [-4.57048808e-01 -2.82101608e-04]
 [-5.21011274e-02 -6.00467203e-02]]
```

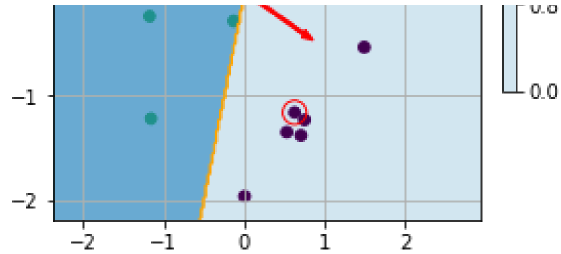
In [4]:

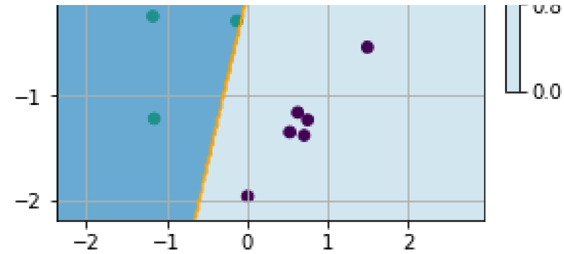
```
nrows = ncols = int(min(3, np.ceil(np.sqrt(N))));
fig, axes = plt.subplots(nrows, ncols, figsize=(12, 12), constrained_layout=True)
for n in np.arange(min(N, nrows * ncols)):
    ax = axes.flat[n]; ax.axis('off'); ax = fig.add_subplot(nrows, ncols, n + 1); ax.grid()
    z = lambda x: np.argmax(W[n+1, :, :] @ x)
    zz = np.apply_along_axis(z, 1, XX)
    ax.contour(xx, yy, zz.reshape(xx.shape), 1, colors='orange', linestyles='solid')
    cp = ax.contourf(xx, yy, zz.reshape(xx.shape), 2, cmap='Blues')
    plt.colorbar(cp, ax=ax, shrink=0.5)
    ax.arrow(0, 0, W[n+1, y[n], 0], W[n+1, y[n], 1], width=.03, facecolor='red', edgecolor='red')
    ax.scatter(*X.T, c=y, s=32); ax.scatter(X[n, 0], X[n, 1], facecolors='none', edgecolors='red', s=150)
```



−2
−1
0
1
2
0.0
0.8
1.6
2.4


0.8
0.0
−1
−2
−2
−1
0
1
2


0.8
0.0
−1
−2
−2
−1
0
1
2


0.8
0.0
−1
−2
−2
−1
0
1
2