T4 Árboles, bosques, bagging y boosting

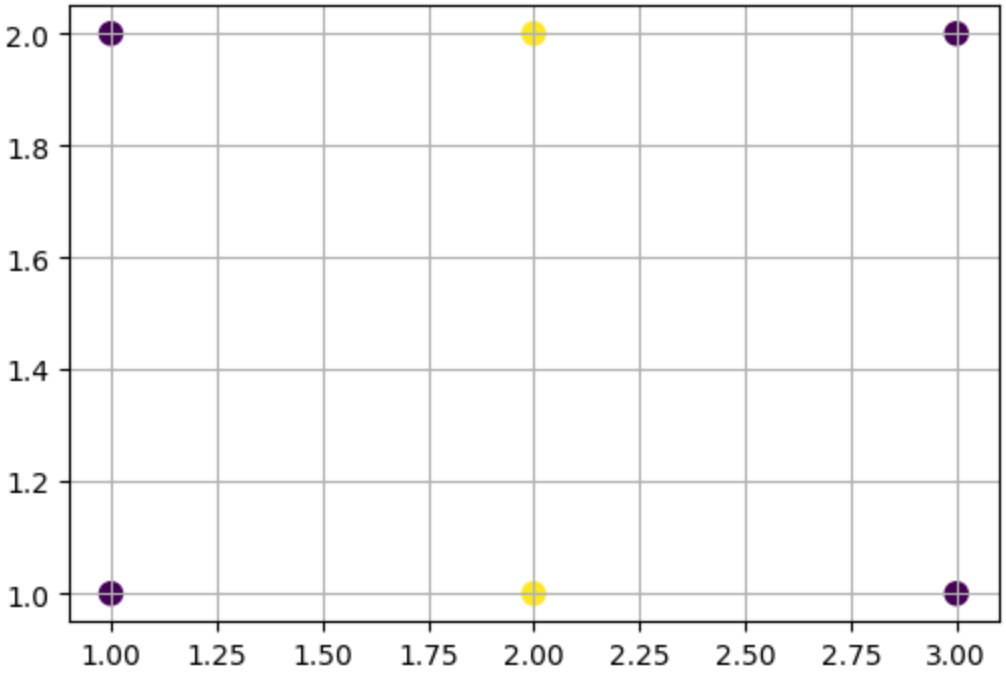
Problemas

1. CART entropía
2. CART Gini
3. CART Gini 3d
4. CART MSE
5. Boosting MSE 1d
6. Boosting MSE 2d
7. Adaboost clf bin 2d
8. Gradient boosting MSE 1d

CART entropía

Sea un problema de clasificación de datos 2d en $C = 2$ clases, $y \in \{1, 2\}$, para el que se está construyendo un árbol de clasificación. El algoritmo de aprendizaje se halla procesando un nodo i cuyo conjunto de datos, $\mathcal{D}_i$, consta de 4 datos de la clase 1, $\{(1, 1)^t, (1, 2)^t, (3, 1)^t, (3, 2)^t\}$, y 2 de la clase 2, $\{(2, 1)^t, (2, 2)^t\}$. Determina la impureza del nodo i, así como un split óptimo del mismo en términos de reducción de impureza. Emplea la entropía para medir la impureza.

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[1, 1], [1, 2], [3, 1], [3, 2], [2, 1], [2, 2]]); y =  np.array([1, 1, 1, 1, 2, 2])
plt.figure(figsize=(6, 4)); plt.grid(); plt.scatter(*X.T, c=y, s=64);
```



Solución:

$$H_i = -\hat{\pi}_{i1}\log_2\hat{\pi}_{i1} - \hat{\pi}_{i2}\log_2\hat{\pi}_{i2} = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} = 0.9183$$

- $j_i = 1, t_i = 1.5$

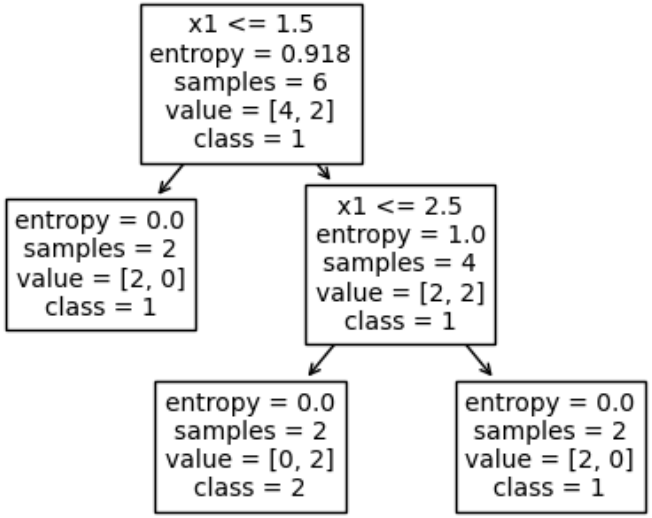
$$H(\mathcal{D}_i^L(1,1.5)) = -1\log_2 1 - 0\log_2 0 = 0$$
$$H(\mathcal{D}_i^R(1,1.5)) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$
$$\Delta(1,1.5) = 0.9183 - \frac{2}{6}\cdot 0 - \frac{4}{6}\cdot 1 = 0.2516$$

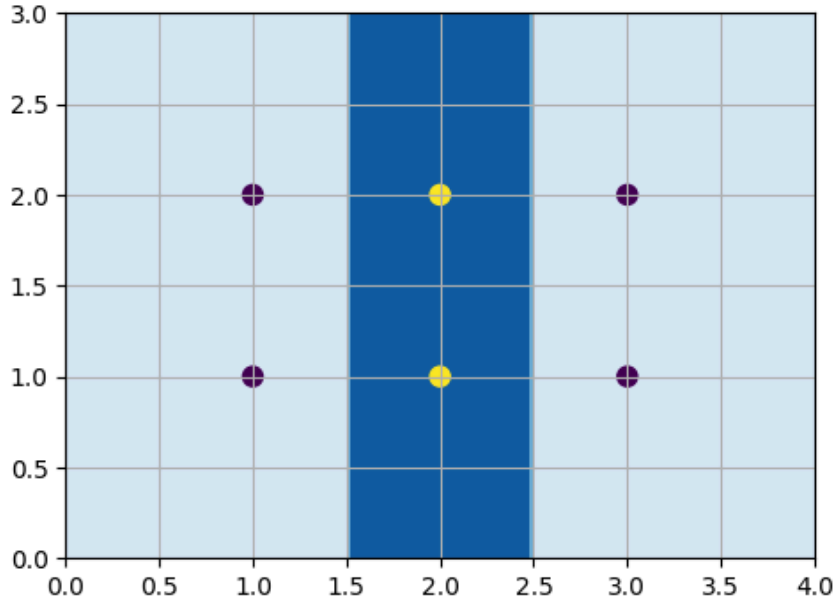
- $j_i = 1, t_i = 2.5 \qquad \Delta(1,2.5) = 0.2516$
- $j_i = 2, t_i = 1.5$

$$H(\mathcal{D}_i^L(2,1.5)) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183$$
$$H(\mathcal{D}_i^R(2,1.5)) = 0.9183$$
$$\Delta(2,1.5) = 0.9183 - \frac{3}{6}\cdot 0.9183 - \frac{3}{6}\cdot 0.9183 = 0$$

- Óptimo: cualquiera de los dos primeros

In [2]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
X = np.array([[1, 1], [1, 2], [3, 1], [3, 2], [2, 1], [2, 2]]); y =  np.array([1, 1, 1, 1, 2, 2])
dt = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=23).fit(X, y)
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_tree(dt, feature_names=list(('x1', 'x2')), class_names=list(('1', '2')), ax=axes[0], fontsize=10);
xx, yy = np.meshgrid(np.linspace(0, 4, num=100), np.linspace(0, 3, num=100))
Z = dt.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape); axes[1].grid()
cp = axes[1].contourf(xx, yy, Z, 2, cmap='Blues'); axes[1].scatter(*X.T, c=y, s=64);
```

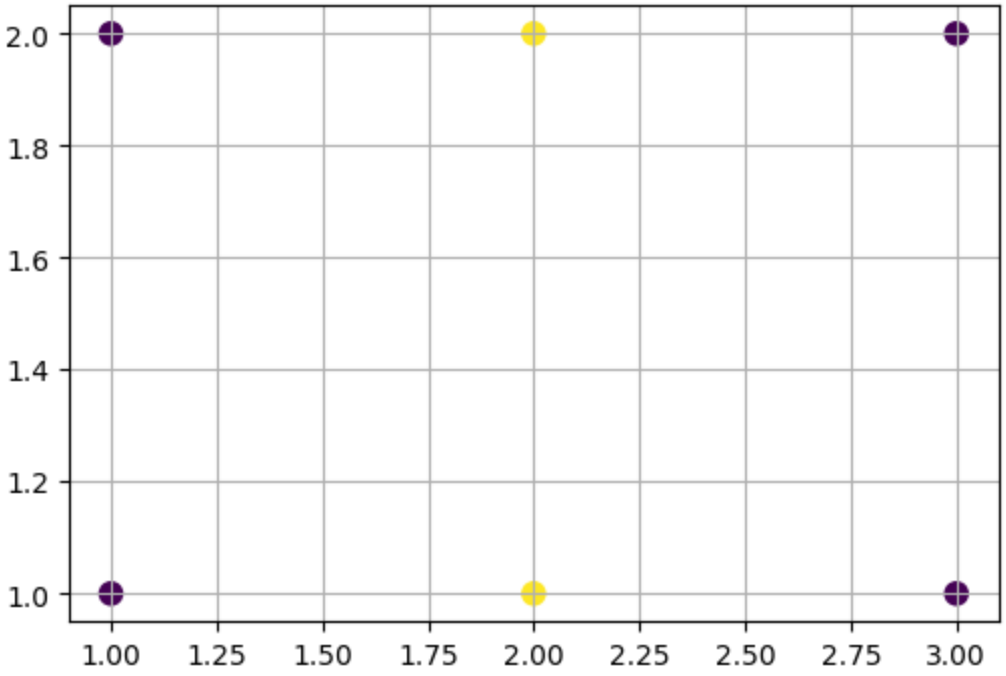




CART Gini

Sea un problema de clasificación de datos 2d en $C = 2$ clases, $y \in \{1, 2\}$, para el que se está construyendo un árbol de clasificación. El algoritmo de aprendizaje se halla procesando un nodo i cuyo conjunto de datos, $\mathcal{D}_i$, consta de 4 datos de la clase 1, $\{(1, 1)^t, (1, 2)^t, (3, 1)^t, (3, 2)^t\}$, y 2 de la clase 2, $\{(2, 1)^t, (2, 2)^t\}$. Determina la impureza del nodo i, así como un split óptimo del mismo en términos de reducción de impureza. Emplea el índice Gini (error de clasificación esperado) para medir la impureza.

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[1, 1], [1, 2], [3, 1], [3, 2], [2, 1], [2, 2]]); y =  np.array([1, 1, 1, 1, 2, 2])
plt.figure(figsize=(6, 4)); plt.grid(); plt.scatter(*X.T, c=y, s=64);
```



Solución:

$$G_i = 1 - \hat{\pi}_{i1}^2 - \hat{\pi}_{i2}^2 = 1 - (4/6)^2 - (2/6)^2 = 1 - 20/36 = 16/36 = 4/9 = 0.44$$

En el eje horizontal (dimensión $d = 1$) tenemos dos posibles splits que dejan dos datos de la clase 1 a un lado y el resto al otro lado, por lo que producen la misma reducción de impureza. Tomemos, por ejemplo, el split basado en el umbral $t = 1.5$. En este caso, dos datos de la clase 1 van al hijo izquierdo y el resto al derecho. Las impurezas de los hijos y reducción de impureza respecto al padre son:

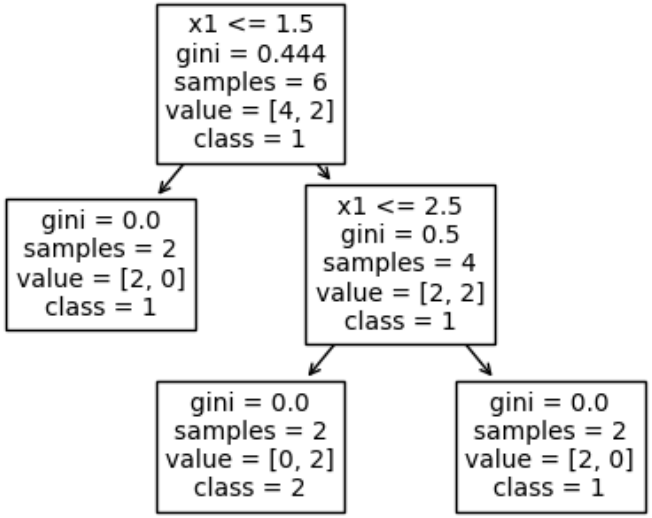
$$\begin{aligned} G(\mathcal{D}_i^L(1, 1.5)) &= 1 - (2/2)^2 - (0/2)^2 = 0 \\ G(\mathcal{D}_i^R(1, 1.5)) &= 1 - (2/4)^2 - (2/4)^2 = 1/2 \\ \Delta &= G_i - 2/6\, G(\mathcal{D}_i^L(1, 1.5)) - 4/6\, G(\mathcal{D}_i^R(1, 1.5)) \\ &= 4/9 - 2/6 \cdot 0 - 4/6 \cdot 1/2 = 1/9 = 0.11 \end{aligned}$$

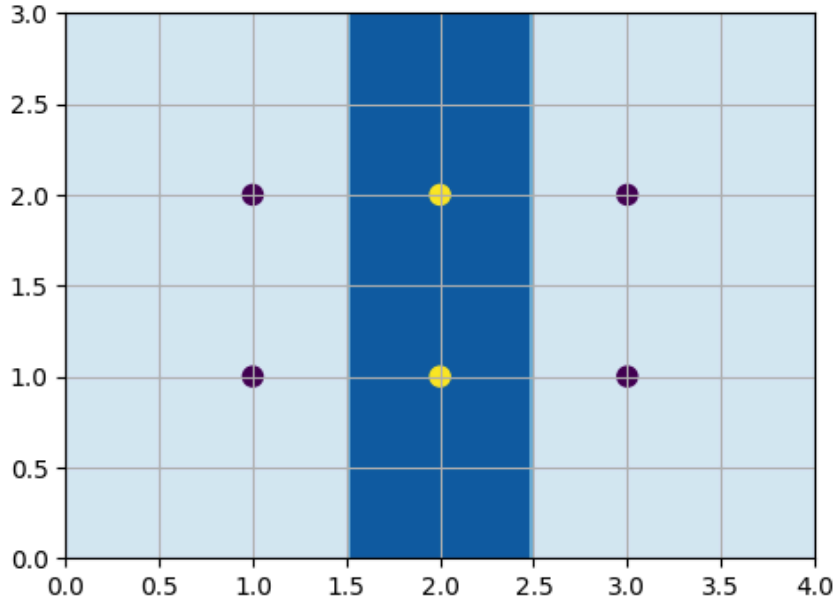
En el eje vertical (dimensión $d = 2$) tenemos un único split que deja ambos hijos con dos datos de la clase 1 y uno de la 2:

$$\begin{aligned} G(\mathcal{D}_i^L(1, 1.5)) &= 1 - (2/3)^2 - (1/3)^2 = 1 - 5/9 = 4/9 \\ G(\mathcal{D}_i^R(1, 1.5)) &= G(\mathcal{D}_i^L(1, 1.5)) \\ \Delta &= G_i - 3/6\, G(\mathcal{D}_i^L(1, 1.5)) - 3/6\, G(\mathcal{D}_i^R(1, 1.5)) \\ &= 4/9 - 3/6 \cdot 4/9 - 3/6 \cdot 4/9 = 0 \end{aligned}$$

Por tanto, un split óptimo consiste en particionar los datos con la primera variable (horizontal) y umbral $t = 1.5$; también sería óptimo emplear el umbral $t = 2.5$.

In [2]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
X = np.array([[1, 1], [1, 2], [3, 1], [3, 2], [2, 1], [2, 2]]); y =  np.array([1, 1, 1, 1, 2, 2])
dt = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=23).fit(X, y)
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_tree(dt, feature_names=list(('x1', 'x2')), class_names=list(('1', '2')), ax=axes[0], fontsize=10);
xx, yy = np.meshgrid(np.linspace(0, 4, num=100), np.linspace(0, 3, num=100))
Z = dt.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape); axes[1].grid()
cp = axes[1].contourf(xx, yy, Z, 2, cmap='Blues'); axes[1].scatter(*X.T, c=y, s=64);
```





CART Gini 3d

Sea un problema de clasificación de datos 3d en $C = 2$ clases, $y \in \{1, 2\}$, para el que se está construyendo un árbol de clasificación. El algoritmo de aprendizaje se halla procesando un nodo i cuyo conjunto de datos, $\mathcal{D}_i$, consta de 2 datos de la clase 1, $\{(1, 1, 2)^t, (1, 2, 1)^t\}$, y 2 de la clase 2, $\{(2, 1, 2)^t, (2, 2, 1)^t\}$. Determina la impureza del nodo i, así como un split óptimo del mismo en términos de reducción de impureza. Emplea el índice Gini (error de clasificación esperado) para medir la impureza.

Solución:

El nodo i tiene dos datos de la clase 1 y otros de la 2, por lo que resulta máximamente impuro:

$$G_i = 1 - \hat{\pi}_{i1}^2 - \hat{\pi}_{i2}^2 = 1 - (2/4)^2 - (2/4)^2 = 1 - 1/2 = 1/2$$

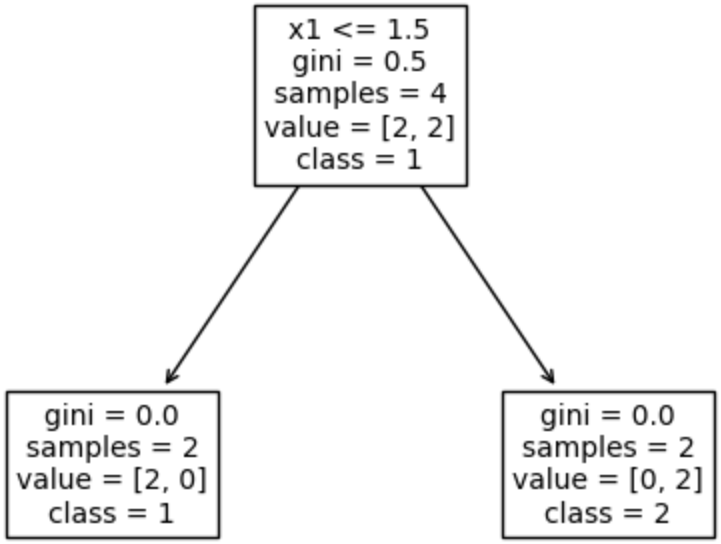
Es fácil determinar los posibles splits del nodo i si tenemos en cuenta que, en las tres dimensiones, los datos están en 1 y 2 únicamente. Por tanto, solo cabe un split por cada dimensión, en 1.5 por ejemplo. El split en la dimensión 1 deja los dos datos de la clase 1 a un lado y los dos datos de la clase 2 al otro; esto es, produce dos nodos hijos puros. Por el contratrio, los splits en las dimensiones 2 y 3 producen hijos con un dato de cada clase; esto es, máximamente impuros, como el nodo i. En definitiva, el split de la dimensión 1 es óptimo y produce la máxima reducción de impureza posible:

$$G(\mathcal{D}_i^L(1, 1.5)) = 1 - (2/2)^2 - (0/2)^2 = 0$$

$$G(\mathcal{D}_i^R(1, 1.5)) = 1 - (0/2)^2 - (2/2)^2 = 0$$

$$\begin{aligned}\Delta &= G_i - 2/4\, G(\mathcal{D}_i^L(1, 1.5)) - 2/4\, G(\mathcal{D}_i^R(1, 1.5)) \\ &= 1/2 - 0 - 0 = 1/2\end{aligned}$$

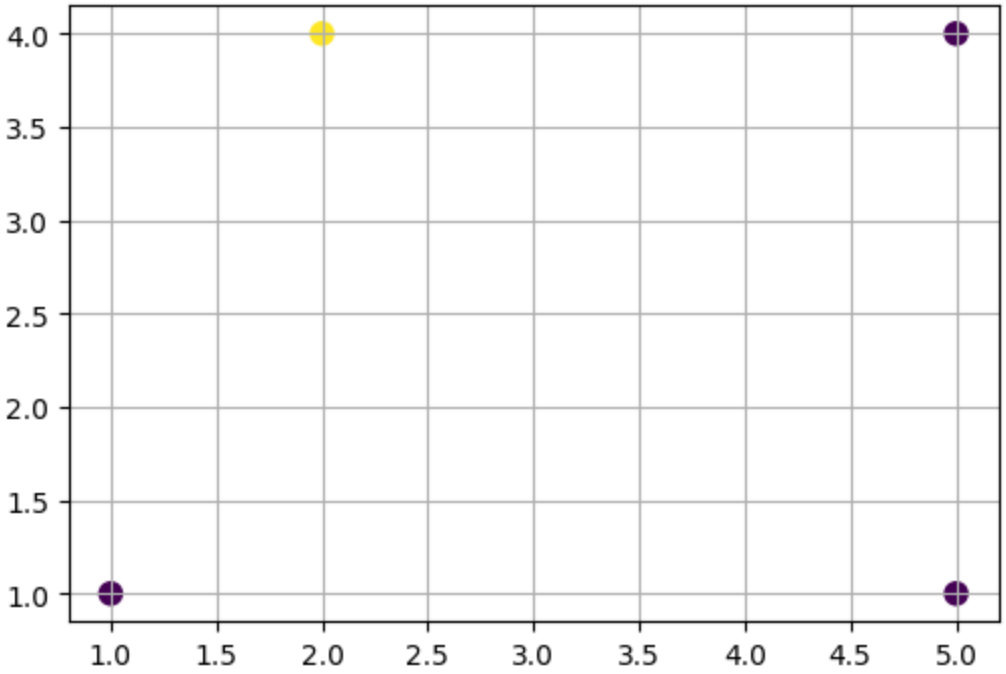
In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
X = np.array([[1, 1, 2], [1, 2, 1], [2, 1, 2], [2, 2, 1]]); y =  np.array([1, 1, 2, 2])
dt = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=23).fit(X, y)
plot_tree(dt, feature_names=list(('x1', 'x2')), class_names=list(('1', '2')), fontsize=10);
```



CART MSE

Sea un problema de regresión de datos 2d en $\mathbb{R}$, para el que se está construyendo un árbol de regresión. El algoritmo de aprendizaje se halla procesando un nodo i de conjunto de datos $\mathcal{D}_i = \{((1,1)^t, 1),\ ((2,4)^t, 2),\ ((5,1)^t, 1),\ ((5,4)^t, 1)\}$. Determina la impureza del nodo i, así como un split óptimo del mismo en términos de reducción de impureza. Emplea el error cuadrático medio (con respecto a la respuesta media del nodo) para medir la impureza.

In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
X = np.array([[1, 1], [2, 4], [5, 1], [5, 4]]); y =  np.array([1, 2, 1, 1])
plt.figure(figsize=(6, 4)); plt.grid(); plt.scatter(*X.T, c=y, s=64);
```



Solución:

$$\hat{y} = \frac{5}{4} = 1.25 \qquad c(\mathcal{D}_i) = \frac{1}{4}(3(1-1.25)^2 + (2-1.25)^2) = 3/16 = 0.1875$$

1. $j_i = 1, t_i = 3.5$

$$c(\mathcal{D}_i^L(1, 3.5)) = \frac{1}{2}((1-1.5)^2 + (2-1.5)^2) = 1/4$$
$$c(\mathcal{D}_i^R(1, 3.5)) = \frac{1}{2}(2(1-1)^2) = 0$$
$$\Delta(1, 3.5) = 3/16 - 2/4 \cdot 1/4 - 2/4 \cdot 0 = 1/16 = 0.0625$$

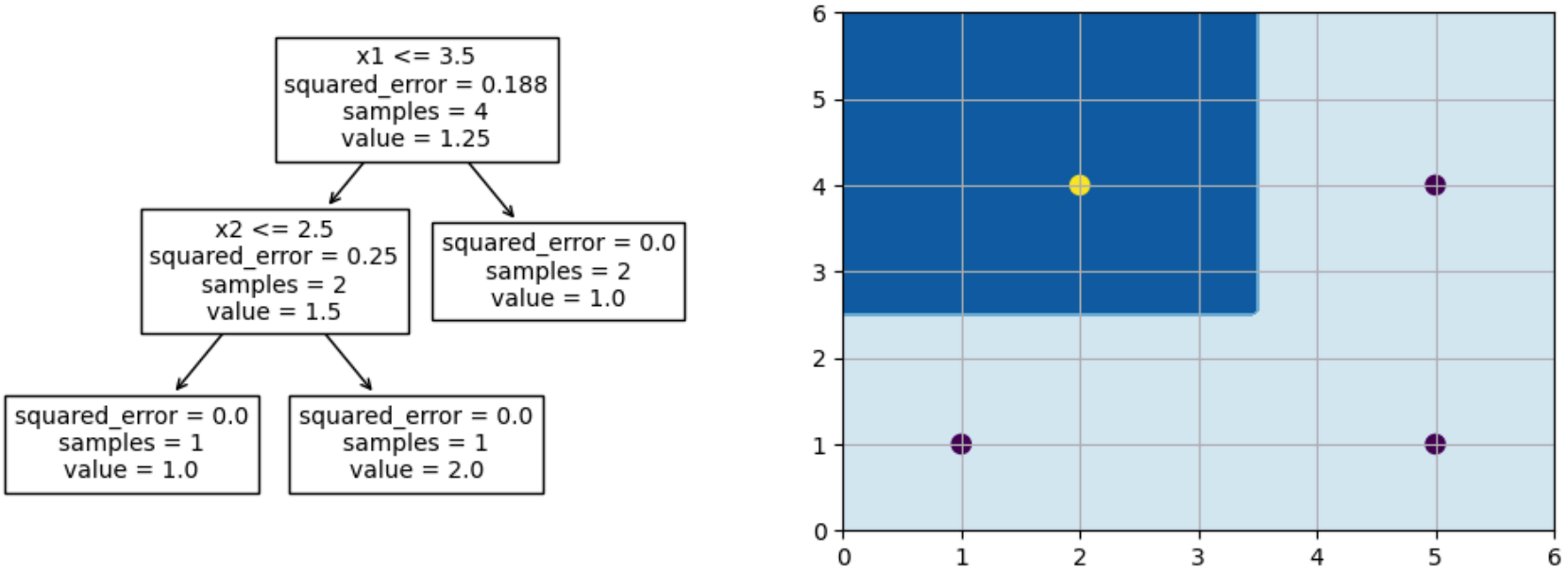
2. $j_i = 1, t_i = 1.5$

$$c(\mathcal{D}_i^L(1, 1.5)) = \frac{1}{1}(1-1)^2 = 0$$
$$c(\mathcal{D}_i^R(1, 1.5)) = \frac{1}{3}((2-4/3)^2 + 2(1-4/3)^2) = 2/9$$
$$\Delta(1, 1.5) = 3/16 - 1/4 \cdot 0 - 3/4 \cdot 2/9 = 0.0208$$

3. $j_i = 2, t_i = 2.5 \rightarrow \Delta = 1/16$

Split óptimo: el 1 o el 3

In [2]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
X = np.array([[1, 1], [2, 4], [5, 1], [5, 4]]); y =  np.array([1, 2, 1, 1])
dt = DecisionTreeRegressor(criterion='squared_error', max_depth=2, random_state=23).fit(X, y)
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_tree(dt, feature_names=list(('x1', 'x2')), ax=axes[0], fontsize=10);
xx, yy = np.meshgrid(np.linspace(0, 6, num=100), np.linspace(0, 6, num=100))
Z = dt.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape); axes[1].grid()
cp = axes[1].contourf(xx, yy, Z, 2, cmap='Blues'); axes[1].scatter(*X.T, c=y, s=64);
```



Boosting MSE 1d

Sea un problema de regresión de datos 1d para el que tenemos $N = 5$ datos de entrenamiento:

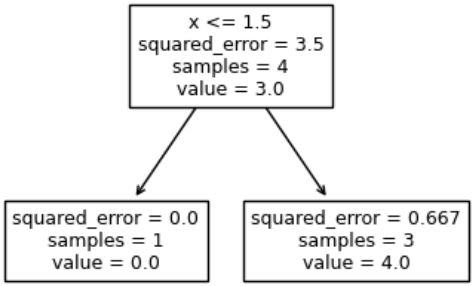
n	1	2	3	4
x_n	1	2	3	4
y	0	4	3	5

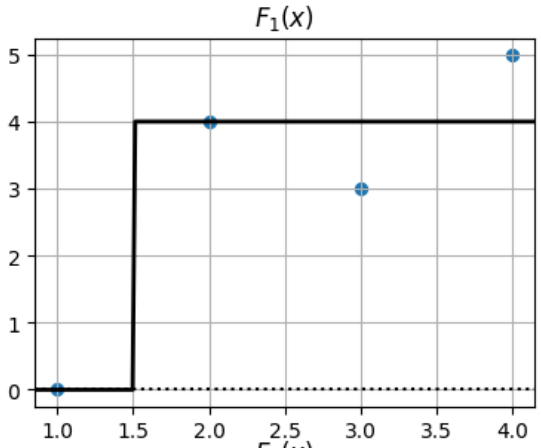
Aplica boosting mínimos cuadrados para ajustar un modelo adaptativo $f(\boldsymbol{x})$ con $M = 2$ modelos base de tipo *stump* (tocón), esto es, árboles de regresión de profundidad uno.

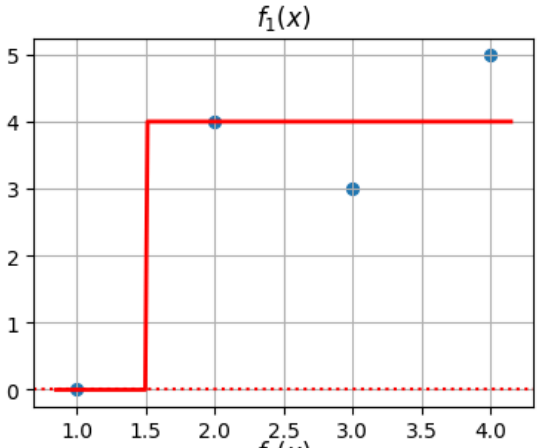
Solución:

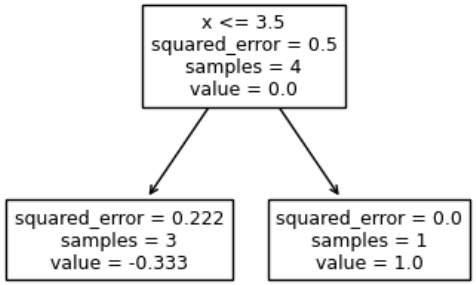
$$F_1(x) = \begin{cases} 0 & \text{si } x \leq 1.5 \\ 4 & \text{en caso contrario} \end{cases} \qquad F_2(x) = \begin{cases} -1/3 & \text{si } x \leq 3.5 \\ 1 & \text{en caso contrario} \end{cases}$$

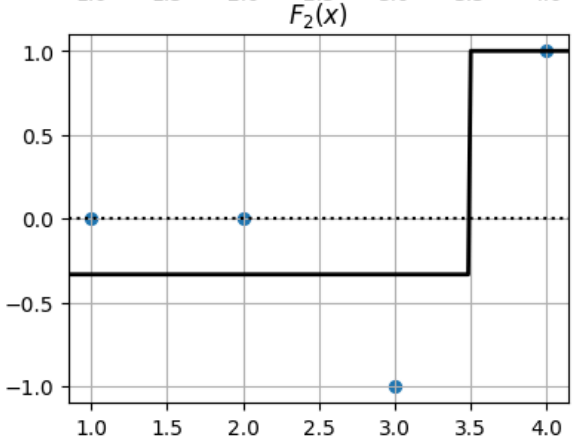
In [4]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
Xy = np.array([[1,0],[2,4],[3,3],[4,5]], dtype=float); X = Xy[:, 0, np.newaxis]; y = Xy[:, 1]
M = 2; fig, axs = plt.subplots(M, 3, figsize=(12, 6)); fig.tight_layout()
res = np.copy(y); tt = []; ax = axs[0, 0]; ax.scatter(X, y)
x_min, x_max = ax.get_xlim(); xx = np.linspace(x_min, x_max, 200)
for m in range(M):
    t = DecisionTreeRegressor(max_depth=1, random_state=42).fit(X, res); tt.append(t)
    plot_tree(t, feature_names=list(('x')), ax=axs[m, 0], fontsize=9)
    ax = axs[m, 1]; ax.grid(); ax.axhline(y=0, color='k', linestyle=':')
    ax.set_title('$F_{}(x)$'.format(m+1)); ax.set_xlim(x_min, x_max); ax.scatter(X, res, s=32)
    res_pred = t.predict(xx.reshape(-1, 1)); ax.plot(xx, res_pred, 'k-', linewidth=2)
    ax = axs[m, 2]; ax.grid(); ax.axhline(y=0, color='r', linestyle=':')
    ax.set_title('$f_{}(x)$'.format(m+1)); ax.scatter(X, y, s=32)
    y_pred = sum(t.predict(xx.reshape(-1, 1)) for t in tt)
    ax.plot(xx, y_pred, 'r-', linewidth=2); res -= t.predict(X)
```

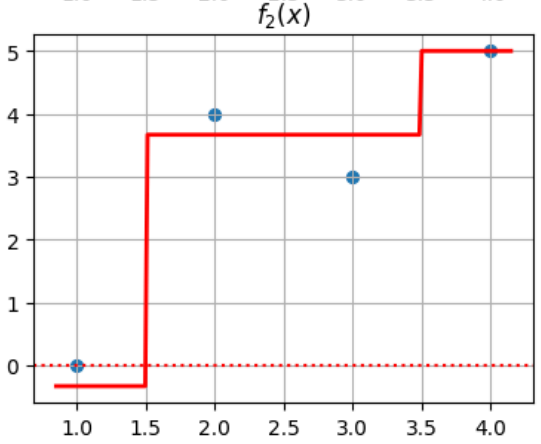

x <= 1.5
squared_error = 3.5
samples = 4
value = 3.0
squared_error = 0.0
samples = 1
value = 0.0
squared_error = 0.667
samples = 3
value = 4.0


$F_1(x)$


$f_1(x)$


x <= 3.5
squared_error = 0.5
samples = 4
value = 0.0
squared_error = 0.222
samples = 3
value = -0.333
squared_error = 0.0
samples = 1
value = 1.0


$F_2(x)$


$f_2(x)$

Boosting MSE 2d

Sea un problema de regresión de datos 2d para el que tenemos $N = 5$ datos de entrenamiento:

n	1	2	3	4	5
$\boldsymbol{x}_n^t$	$(2, 3)$	$(4, 4)$	$(6, 6)$	$(8, 2)$	$(8, 7)$
$\tilde{y}$	1	-1	1	-2	-1

Se ha empleado boosting mínimos cuadrados para ajustar un modelo adaptativo $f(\boldsymbol{x})$ con $M = 3$ modelos base de tipo *stump* (tocón), esto es, árboles de regresión de profundidad uno. Cada modelo base se caracteriza por el split aplicado y las respuestas de cada una de sus dos hojas:

$$F_1(\boldsymbol{x}) = \begin{cases} 1/3 & \text{si } x_1 \leq 7 \\ -1.5 & \text{en caso contrario} \end{cases} \qquad F_2(\boldsymbol{x}) = \begin{cases} -0.389 & \text{si } x_2 \leq 5 \\ 0.583 & \text{en caso contrario} \end{cases} \qquad F_3(\boldsymbol{x}) = \begin{cases} 1.056 & \text{si } x \\ -0.264 & \text{en} \end{cases}$$

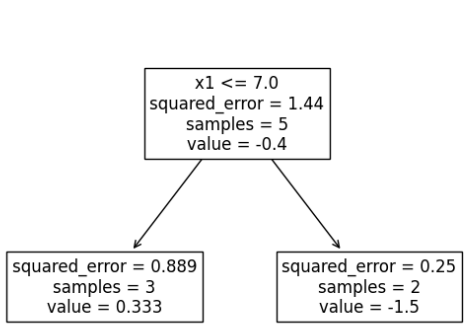
Se pide:

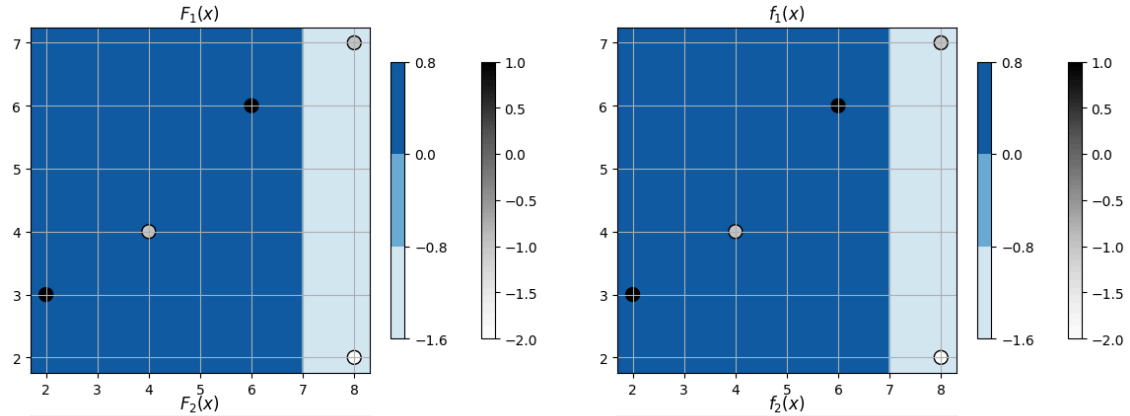
1. Determina $f(4, 6)$.
2. Representa gráficamente $f(\boldsymbol{x})$.

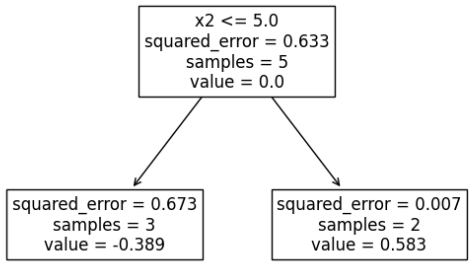
Solución:

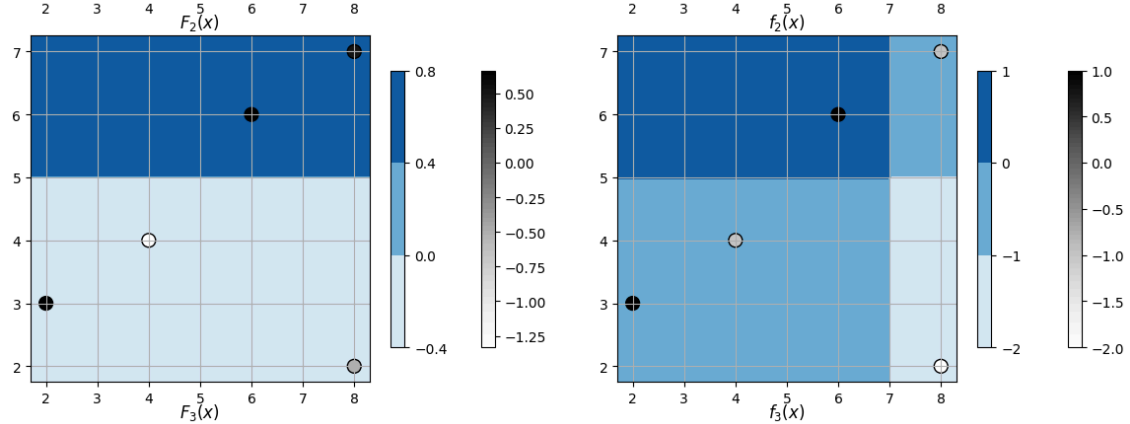
$$f(4,6) = F_1(4,6) + F_2(4,6) + F_3(4,6) = 1/3 + 0.583 - 0.264 = 0.6523$$

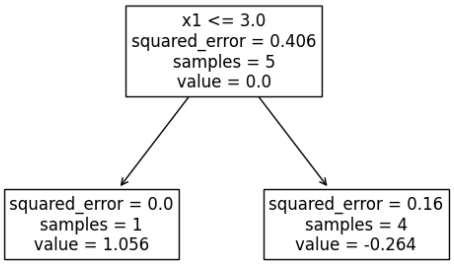
In [2]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
Xy = np.array([[2,3,1],[4,4,-1],[6,6,1],[8,7,-1],[8,2,-2]], dtype=float)
X = Xy[:, :2]; y = Xy[:, 2]; M = 3
fig, axs = plt.subplots(M, 3, figsize=(6*M, 12)); fig.tight_layout()
res = np.copy(y); tt = []; ax = axs[0, 0]; ax.scatter(*X.T, c=y, s=64)
x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
X_test = np.c_[xx.ravel(), yy.ravel()]
for m in range(M):
    t = DecisionTreeRegressor(max_depth=1, random_state=23).fit(X, res); tt.append(t)
    plot_tree(t, feature_names=list(('x1', 'x2')), ax=axs[m, 0], fontsize=12)
    ax = axs[m, 1]; ax.grid(); ax.set_title('$F_{}(x)$'.format(m+1))
    F = t.predict(X_test).reshape(xx.shape)
    cp = ax.contourf(xx, yy, F, 2, cmap='Blues')
    sp = ax.scatter(*X.T, c=res.reshape(1, -1), cmap='Greys', edgecolors='black', s=100)
    plt.colorbar(sp, ax=ax, shrink=0.8); plt.colorbar(cp, ax=ax, shrink=0.8)
    ax = axs[m, 2]; ax.grid(); ax.set_title('$f_{}(x)$'.format(m+1))
    f = sum(t.predict(X_test) for t in tt).reshape(xx.shape)
    cp = ax.contourf(xx, yy, f, 2, cmap='Blues')
    sp = ax.scatter(*X.T, c=y, cmap='Greys', edgecolors='black', s=100)
    plt.colorbar(sp, ax=ax, shrink=0.8); plt.colorbar(cp, ax=ax, shrink=0.8)
    res -= t.predict(X)
```

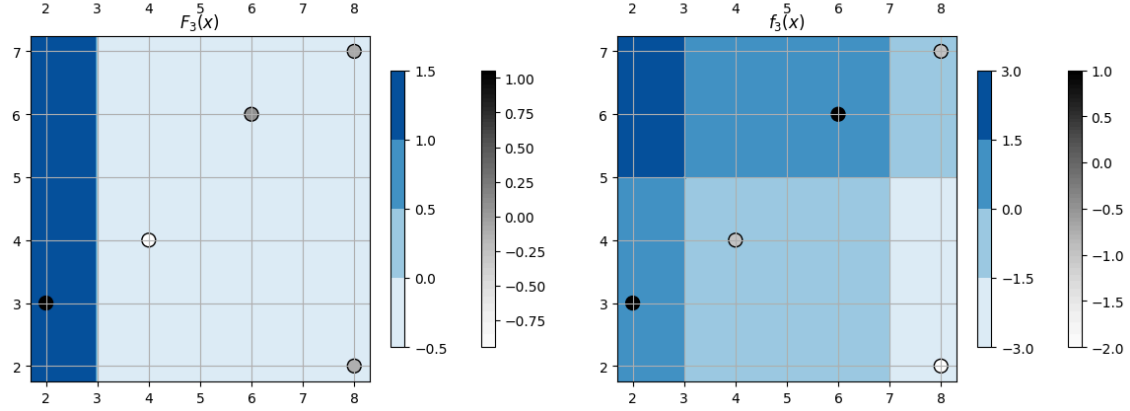

x1 <= 7.0
squared_error = 1.44
samples = 5
value = -0.4
squared_error = 0.889
samples = 3
value = 0.333
squared_error = 0.25
samples = 2
value = -1.5


$F_1(x)$
$f_1(x)$


x2 <= 5.0
squared_error = 0.633
samples = 5
value = 0.0
squared_error = 0.673
samples = 3
value = -0.389
squared_error = 0.007
samples = 2
value = 0.583


$F_2(x)$
$f_2(x)$


x1 <= 3.0
squared_error = 0.406
samples = 5
value = 0.0
squared_error = 0.0
samples = 1
value = 1.056
squared_error = 0.16
samples = 4
value = -0.264


$F_3(x)$
$f_3(x)$

Adaboost clf bin 2d

Sea un problema de clasificación de datos 2d en $C = 2$ clases, $y \in \{-1, +1\}$, para el que tenemos $N = 5$ datos de entrenamiento:

n	1	2	3	4	5
$\boldsymbol{x}_n^t$	$(2,3)$	$(4,4)$	$(6,6)$	$(8,2)$	$(8,7)$
$\tilde{y}$	1	-1	1	-1	-1

Tras aplicar Adaboost para ajustar un modelo adaptativo $f(\boldsymbol{x})$ con $M = 3$ modelos base de tipo *stump* (tocón), se han obtenido los siguientes árboles:

$$F_1(\boldsymbol{x}) = 2\mathbb{I}(x_1 \leq 7) - 1 \qquad F_2(\boldsymbol{x}) = 2\mathbb{I}(x_1 \leq 3) - 1 \qquad F_3(\boldsymbol{x}) = 2\mathbb{I}(x_2 \leq 5) - 1$$

Se pide:

1. Para $m \in \{1, 2, 3\}$, halla los pesos de los datos, $w_{nm}\,(n = 1, \ldots, 5)$, la clasificación de los mismos según F_m, el error ponderado err_m y el peso de F_m en el ensamble f, β_m.
2. Clasifica $\boldsymbol{x}^t = (4, 6)$ con el ensamble de los dos primeros modelos base y con el ensamble completo.

Solución:

- Iteración $m = 1$: los pesos iniciales son $w_{n1} = 1/N$

n	1	2	3	4	5
w_{n1}	0.2	0.2	0.2	0.2	0.2
$F_1(\boldsymbol{x}_n)$	1	1	1	-1	-1

$$\text{err}_1 = 0.2 \quad \rightarrow \quad \beta_1 = \frac{1}{2}\log\frac{0.8}{0.2} = 0.6931$$

- Iteración $m = 2$: solo se corrige el peso del segundo dato, $w_{22} = w_{21}\exp(2\beta_1) = 0.8$

n	1	2	3	4	5
w_{n2}	0.2	0.8	0.2	0.2	0.2
$F_2(\boldsymbol{x}_n)$	1	-1	-1	-1	-1

$$\text{err}_2 = \frac{1}{1.6}0.2 = 0.125 \quad \rightarrow \quad \beta_2 = \frac{1}{2}\log\frac{0.875}{0.125} = 0.9730$$

- Iteración $m = 3$: solo se corrige el peso del tercer dato, $w_{33} = w_{32}\exp(2\beta_2) = 1.4$

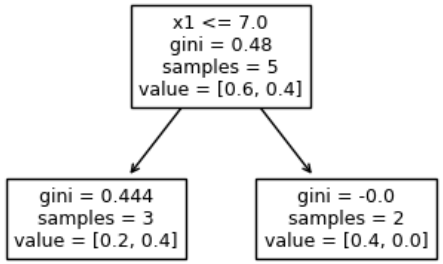
n	1	2	3	4	5
w_{n3}	0.2	0.8	1.4	0.2	0.2
$F_3(\boldsymbol{x}_n)$	1	1	-1	1	-1

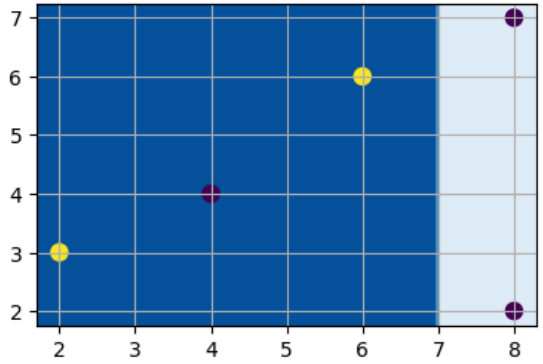
$$\text{err}_3 = \frac{1}{2.8}(0.8 + 1.4 + 0.2) = \frac{2.4}{2.8} = 0.8571 \quad \rightarrow \quad \beta_3 = \frac{1}{2}\log\frac{1 - 0.8571}{0.8571} = -0.8959$$

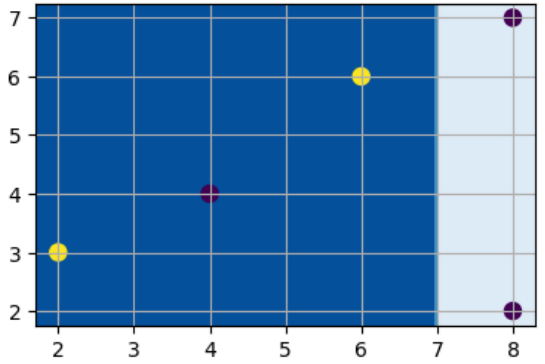
- Clasificación de $\boldsymbol{x}^t = (4, 6)$: $f(\boldsymbol{x}) = \text{sgn}(\beta_1 F_1(\boldsymbol{x}) + \beta_2 F_2(\boldsymbol{x}) + \beta_3 F_3(\boldsymbol{x}))$

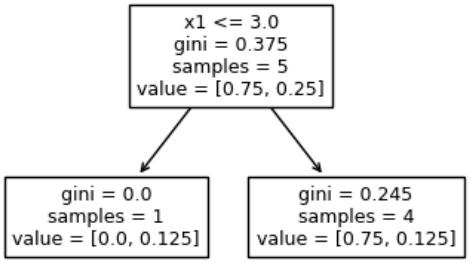
$$f(\boldsymbol{x}) = \text{sgn}(0.6931 \cdot 1 + 0.9730 \cdot (-1) + (-0.8959) \cdot (-1)) = \text{sgn}(0.6160) = 1$$

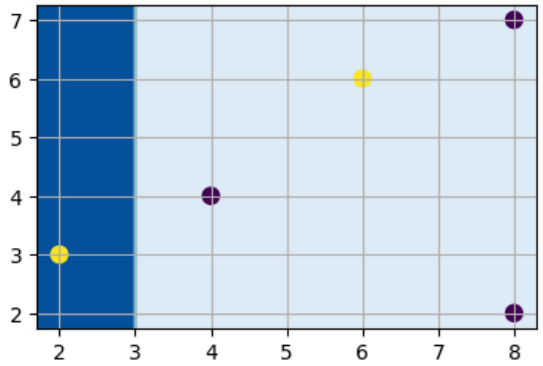
In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import plot_tree; from sklearn.ensemble import AdaBoostClassifier
Xy = np.array([[2,3,1],[4,4,-1],[6,6,1],[8,7,-1],[8,2,-1]]); X = Xy[:, :2]; y = Xy[:, 2]
M = 3; fig, axs = plt.subplots(M, 3, figsize=(4*M, 8)); fig.tight_layout()
f = AdaBoostClassifier(n_estimators=M, algorithm="SAMME", random_state=23).fit(X, y)
ax = axs[0, 0]; ax.scatter(*X.T, c=y, s=64); x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
y_test = np.c_[xx.ravel(), yy.ravel()]; y_pred = f.staged_predict(y_test)
for m, y_pred in enumerate(y_pred):
    plot_tree(f.estimators_[m], feature_names=list(('x1', 'x2')), ax=axs[m, 0], fontsize=9)
    ax = axs[m, 1]; ax.grid(); Z = f.estimators_[m].predict(y_test).reshape(xx.shape)
    cp = ax.contourf(xx, yy, Z, 2, cmap='Blues'); ax.scatter(*X.T, c=y, s=64)
    ax = axs[m, 2]; ax.grid(); Z = y_pred.reshape(xx.shape)
    cp = ax.contourf(xx, yy, Z, 2, cmap='Blues'); ax.scatter(*X.T, c=y, s=64)
```

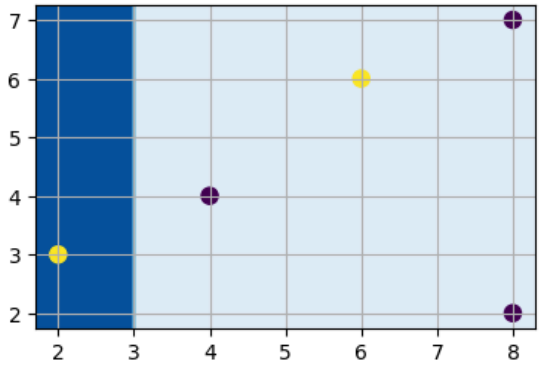

x1 <= 7.0
gini = 0.48
samples = 5
value = [0.6, 0.4]
gini = 0.444
samples = 3
value = [0.2, 0.4]
gini = -0.0
samples = 2
value = [0.4, 0.0]

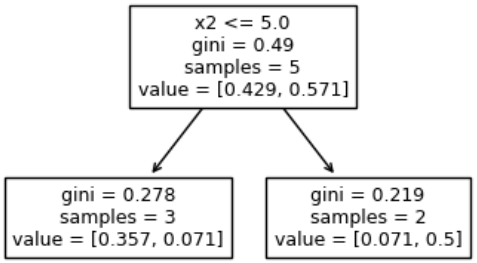


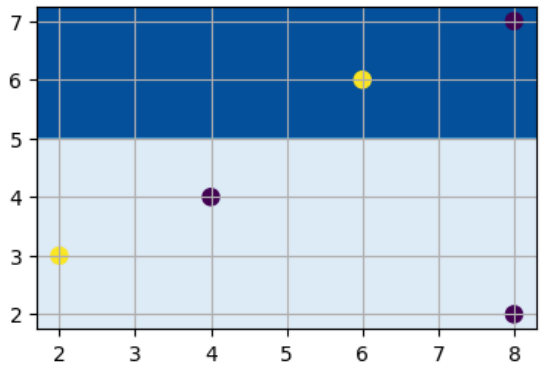


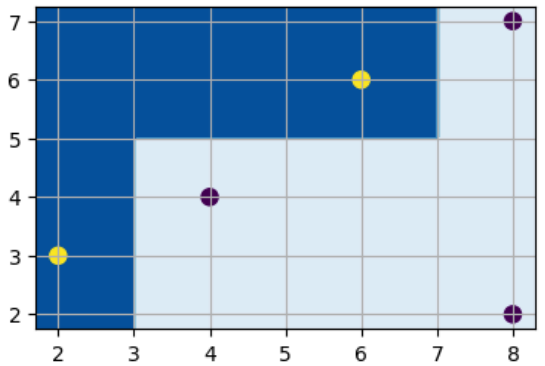

x1 <= 3.0
gini = 0.375
samples = 5
value = [0.75, 0.25]
gini = 0.0
samples = 1
value = [0.0, 0.125]
gini = 0.245
samples = 4
value = [0.75, 0.125]






x2 <= 5.0
gini = 0.49
samples = 5
value = [0.429, 0.571]
gini = 0.278
samples = 3
value = [0.357, 0.071]
gini = 0.219
samples = 2
value = [0.071, 0.5]





Gradient boosting MSE 1d

Sea un problema de regresión de datos 1d para el que tenemos $N = 5$ datos de entrenamiento:

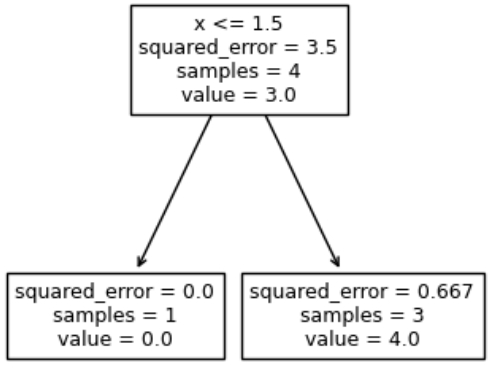
n	1	2	3	4
x_n	1	2	3	4
y	0	4	3	5

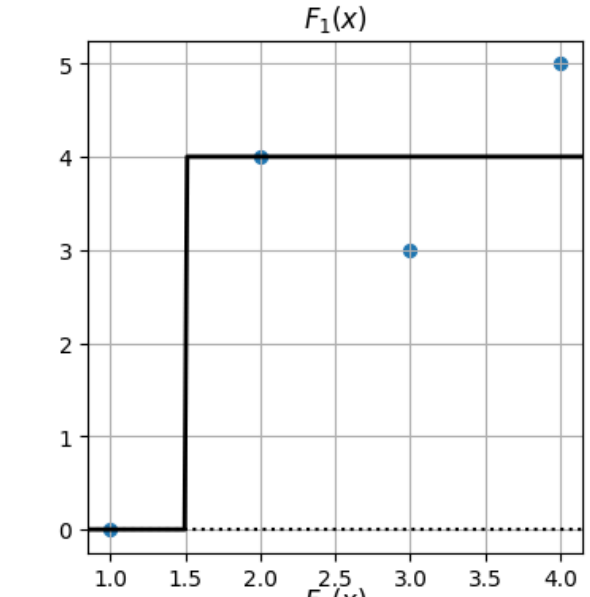
Aplica gradient boosting para ajustar un modelo adaptativo $f(\boldsymbol{x})$ con $M = 2$ modelos base de tipo *stump* (tocón), esto es, árboles de regresión de profundidad uno.

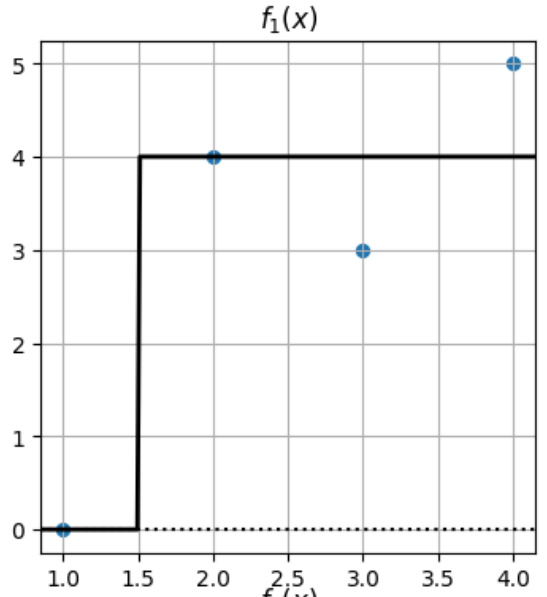
Solución: con modelo inicial nulo por simplicidad

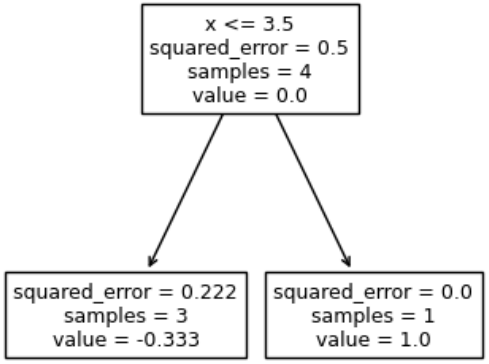
$$f_0(x) = 0 \qquad F_1(x) = \begin{cases} -3 & \text{si } x \leq 1.5 \\ 1 & \text{en caso contrario} \end{cases} \qquad F_2(x) = \begin{cases} -1/3 & \text{si } x \leq 3.5 \\ 1 & \text{en caso contrario} \end{cases}$$

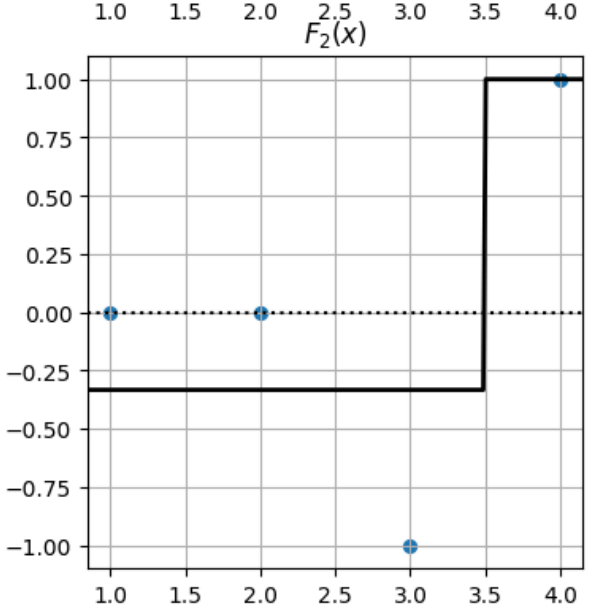
In [1]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import plot_tree; from sklearn.ensemble import GradientBoostingRegressor
Xy = np.array([[1,0],[2,4],[3,3],[4,5]], dtype=float); X = Xy[:, 0]; y = Xy[:, 1]; M = 2
f = GradientBoostingRegressor(loss='squared_error', learning_rate=1.0, criterion='squared_error',
    n_estimators=M, max_depth=1, init='zero', random_state=23).fit(X.reshape(-1, 1), y)
fig, axs = plt.subplots(M, 3, figsize=(3.75*3, 3.75*M)); fig.tight_layout(); y_res = np.copy(y)
for m in range(M):
    ax = axs[m, 0]; ax.scatter(X, y); x_min, x_max = ax.get_xlim(); xx = np.linspace(x_min, x_max, 200)
    plot_tree(f.estimators_[m, 0], feature_names=list(('x')), ax=ax, fontsize=9)
    ax = axs[m, 1]; ax.set_xlim(x_min, x_max); ax.grid(); ax.axhline(y=0, color='k', linestyle=':')
    ax.set_title('$F_{}(x)$'.format(m+1)); ax.set_xlim(x_min, x_max); ax.scatter(X, y_res, s=32)
    yy = f.estimators_[m, 0].predict(xx.reshape(-1, 1)); ax.plot(xx, yy, 'k-', linewidth=2);
    y_res_pred = f.estimators_[m, 0].predict(X.reshape(-1, 1)); y_res -= y_res_pred
    ax = axs[m, 2]; ax.set_xlim(x_min, x_max); ax.grid(); ax.axhline(y=0, color='k', linestyle=':')
    ax.set_title('$f_{}(x)$'.format(m+1)); ax.scatter(X, y, s=32)
    yy = sum(f.estimators_[i, 0].predict(xx.reshape(-1, 1)) for i in range(m+1))
    ax.plot(xx, yy, 'k-', linewidth=2);
```

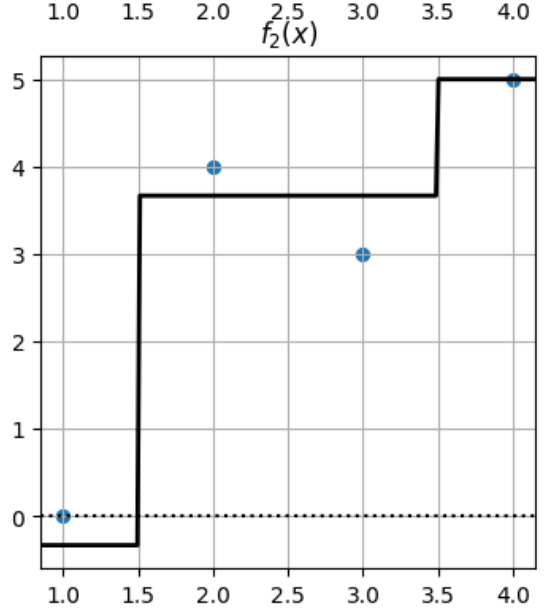

x <= 1.5
squared_error = 3.5
samples = 4
value = 3.0
squared_error = 0.0
samples = 1
value = 0.0
squared_error = 0.667
samples = 3
value = 4.0


$F_1(x)$


$f_1(x)$


x <= 3.5
squared_error = 0.5
samples = 4
value = 0.0
squared_error = 0.222
samples = 3
value = -0.333
squared_error = 0.0
samples = 1
value = 1.0


$F_2(x)$


$f_2(x)$

Gradient boosting clf bin 2d

Sea un problema de clasificación de datos 2d en $C = 2$ clases, $y \in \{-1, +1\}$, para el que tenemos $N = 5$ datos de entrenamiento:

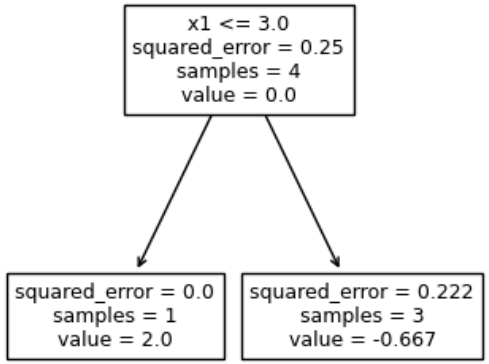
n	1	2	3	4	5
$\boldsymbol{x}_n^t$	$(2, 3)$	$(4, 4)$	$(6, 6)$	$(8, 2)$	$(8, 7)$
$\tilde{y}$	1	-1	1	-1	-1

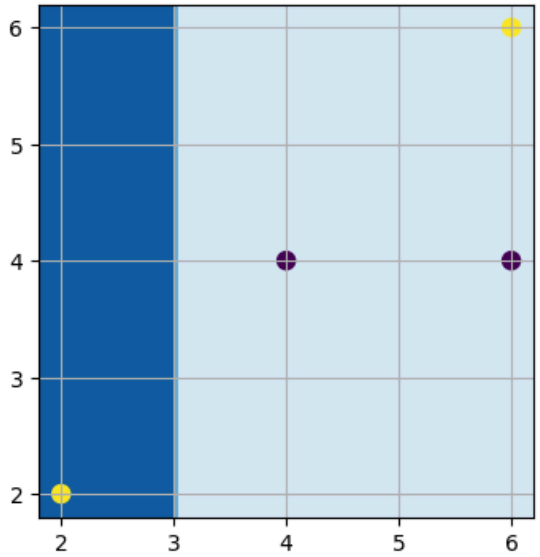
Se pide:

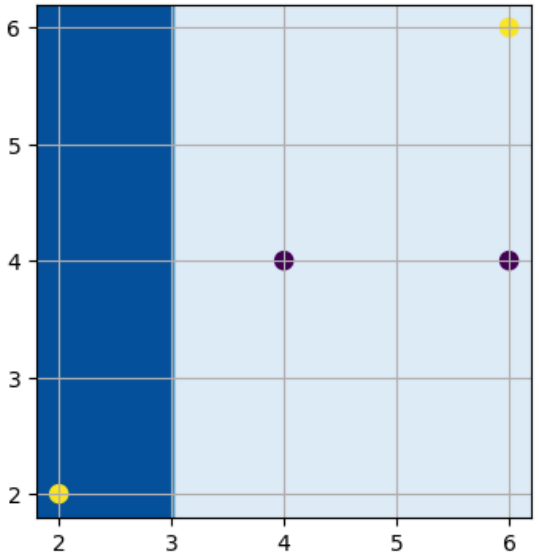
1. Aplica gradient boosting para ajustar un modelo adaptativo $f(\boldsymbol{x})$ con $M = 2$ modelos base de tipo *stump*.
2. Clasifica $\boldsymbol{x}^t = (4, 6)$ con el ensamble ajustado en el paso anterior.

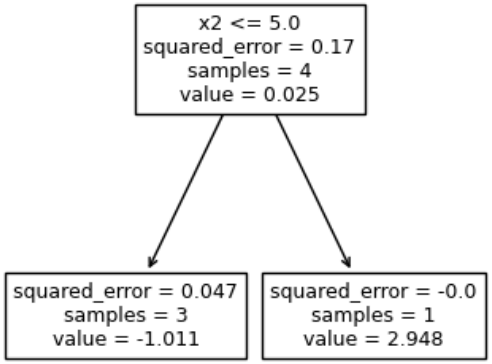
Solución:

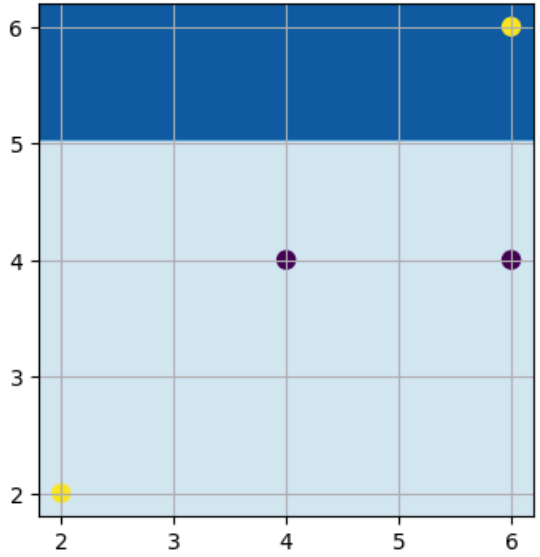
In [29]:
```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.tree import plot_tree; from sklearn.ensemble import GradientBoostingClassifier
Xy = np.array([[2,2,1],[4,4,-1],[6,6,1],[6,4,-1]]); X = Xy[:, :2]; y = Xy[:, 2]
M = 2; fig, axs = plt.subplots(M, 3, figsize=(3.75*3, 3.75*M)); fig.tight_layout()
f = GradientBoostingClassifier(loss='log_loss', learning_rate=1.0, criterion='squared_error',
    n_estimators=M, max_depth=1, init='zero', random_state=23).fit(X, y)
ax = axs[0, 0]; ax.scatter(*X.T, c=y, s=64); x_min, x_max = ax.get_xlim(); y_min, y_max = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
y_test = np.c_[xx.ravel(), yy.ravel()]; y_pred = f.staged_predict(y_test)
for m, y_pred in enumerate(y_pred):
    plot_tree(f.estimators_[m, 0], feature_names=list(('x1', 'x2')), ax=axs[m, 0], fontsize=9)ax.set_xtick
    Z = y_pred.reshape(xx.shape)
    cp = ax.contourf(xx, yy, Z, 2, cmap='Blues'); ax.scatter(*X.T, c=y, s=64)
```

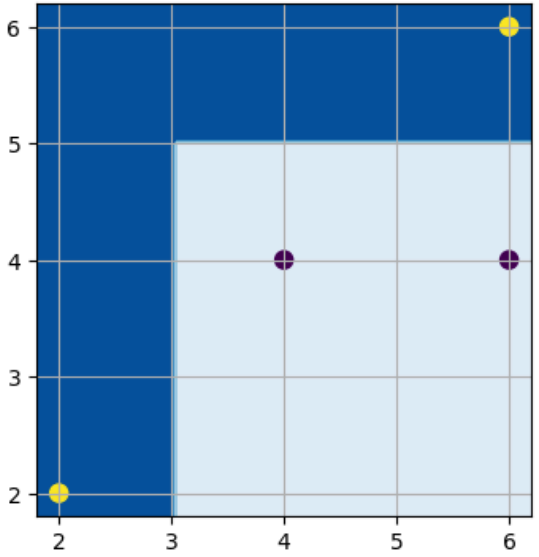

x1 <= 3.0
squared_error = 0.25
samples = 4
value = 0.0
squared_error = 0.0
samples = 1
value = 2.0
squared_error = 0.222
samples = 3
value = -0.667


6
5
4
3
2
2
3
4
5
6


6
5
4
3
2
2
3
4
5
6


x2 <= 5.0
squared_error = 0.17
samples = 4
value = 0.025
squared_error = 0.047
samples = 3
value = -1.011
squared_error = -0.0
samples = 1
value = 2.948


6
5
4
3
2
2
3
4
5
6


6
5
4
3
2
2
3
4
5
6

T4 Árboles, bosques, bagging y boosting

Cuestiones

1. CART
2. Gradient boosting
3. Comparación

CART: Los árboles presentan una serie de ventajas e incovenientes que cabe tener presentes para su uso. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. Son insensibles a transformaciones monótonas de las entradas ya que los puntos de split se basan en la ordenación de los datos, por lo que es necesario estandarizarlos.
2. Son de ajuste rápido y fácil escalado a muchos datos.
3. Son inestables, esto es, pequeños cambios en los datos de entrada pueden conducir a grandes cambios en la estructura del árbol.
4. Todas son correctas.

CART: Los árboles presentan una serie de ventajas e incovenientes que cabe tener presentes para su uso. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. Son insensibles a transformaciones monótonas de las entradas ya que los puntos de split se basan en la ordenación de los datos, por lo que es necesario estandarizarlos.
2. Son de ajuste rápido y fácil escalado a muchos datos.
3. Son inestables, esto es, pequeños cambios en los datos de entrada pueden conducir a grandes cambios en la estructura del árbol.
4. Todas son correctas.

Solución:

La 1 es incorrecta. Al revés de lo que dice, no es necesario estandarizar los datos porque los árboles son insensibles a transformaciones monótonas de las entradas.

Gradient boosting: Gradient boosting es una técnica **forward stagewise additive modeling (FSAM),** pero abandona la idea de reponderar datos iterativamente y replantea FSAM como descenso por gradiente para un problema de búsqueda en un espacio funcional, $\hat{\boldsymbol{f}} = \operatorname{argmin}_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f})$. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. El paso m halla $\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \beta_m \boldsymbol{g}_m$, donde $\boldsymbol{g}_m$ es el gradiente de $\mathcal{L}(\boldsymbol{f})$ en $\boldsymbol{f} = \boldsymbol{f}_{m-1}$ y β_m se obtiene por búsqueda lineal.
2. $\boldsymbol{f}_m$ se obtiene añadiendo a $\boldsymbol{f}_{m-1}$ un modelo base $F_m(\boldsymbol{x})$ ajustado por mínimos cuadrados al residuo del gradiente.
3. β_m suele sustituirse por un factor de reducción $0 < \nu \leq 1$ que permite regularizar y, así, $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu F_m(\boldsymbol{x})$.
4. Todas son correctas.

Gradient boosting: Gradient boosting es una técnica **forward stagewise additive modeling (FSAM),** pero abandona la idea de reponderar datos iterativamente y replantea FSAM como descenso por gradiente para un problema de búsqueda en un espacio funcional, $\hat{\boldsymbol{f}} = \operatorname{argmin}_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f})$. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. El paso m halla $\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \beta_m \boldsymbol{g}_m$, donde $\boldsymbol{g}_m$ es el gradiente de $\mathcal{L}(\boldsymbol{f})$ en $\boldsymbol{f} = \boldsymbol{f}_{m-1}$ y β_m se obtiene por búsqueda lineal.
2. $\boldsymbol{f}_m$ se obtiene añadiendo a $\boldsymbol{f}_{m-1}$ un modelo base $F_m(\boldsymbol{x})$ ajustado por mínimos cuadrados al residuo del gradiente.
3. β_m suele sustituirse por un factor de reducción $0 < \nu \leq 1$ que permite regularizar y, así, $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu F_m(\boldsymbol{x})$.
4. Todas son correctas.

Solución:

La 4.

Comparación: Los árboles constituyen un estimador de alta varianza ya que pequeñas perturbaciones de los datos de entrenamiento resultan en predicciones muy distintas. El aprendizaje de ensambles reduce la varianza de los árboles promediando $|\mathcal{M}|$ modelos base $\{f_m\}$, $f(y \mid \boldsymbol{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y \mid \boldsymbol{x})$. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. En regresión, el promediado suele presentar un sesgo similar al de los modelos base, pero mejor precisión por la menor varianza.
2. En clasificación se usa el voto mayoritario o método comité, que incrementa la precisión de los modelos base, especialmente cuando sus errores de predicción no se hallan correlados.
3. Bagging, bosques aleatorios y gradient tree boosting aprenden ensambles de árboles con el fin de reducir la varianza. Además, bosques aleatorios y gradient tree boosting adaptan el ajuste de cada modelo base teniendo en cuenta los modelos base ajustados previamente. De esta manera, no solo consiguen reducir la varianza, sino que también reducen el sesgo del ensamble resultante.
4. Todas son correctas.

Comparación: Los árboles constituyen un estimador de alta varianza ya que pequeñas perturbaciones de los datos de entrenamiento resultan en predicciones muy distintas. El aprendizaje de ensambles reduce la varianza de los árboles promediando $|\mathcal{M}|$ modelos base $\{f_m\}$, $f(y \mid \boldsymbol{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y \mid \boldsymbol{x})$. Indica la respuesta incorrecta (o escoge la última opción si las tres primeras son correctas).

1. En regresión, el promediado suele presentar un sesgo similar al de los modelos base, pero mejor precisión por la menor varianza.
2. En clasificación se usa el voto mayoritario o método comité, que incrementa la precisión de los modelos base, especialmente cuando sus errores de predicción no se hallan correlados.
3. Bagging, bosques aleatorios y gradient tree boosting aprenden ensambles de árboles con el fin de reducir la varianza. Además, bosques aleatorios y gradient tree boosting adaptan el ajuste de cada modelo base teniendo en cuenta los modelos base ajustados previamente. De esta manera, no solo consiguen reducir la varianza, sino que también reducen el sesgo del ensamble resultante.
4. Todas son correctas.

Solución:

La 3; bosques aleatorios no reduce el sesgo.