In [1]:

```
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

Дерева рішень

Тренування та візуалізація

In [2]:

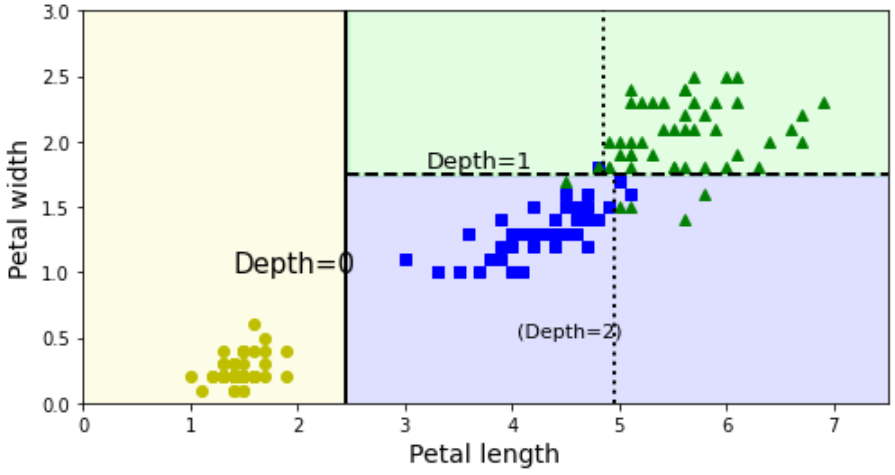
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
```

Out[2]:

```
DecisionTreeClassifier(max_depth=2, random_state=42)
```

In [3]:

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True, legend=False, plot_training=Tru
e):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if not iris:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris setosa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris versicolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris virginica")
        plt.axis(axes)
    if iris:
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
        plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
    if legend:
        plt.legend(loc="lower right", fontsize=14)

plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf, X, y)
plt.plot([2.45, 2.45], [0, 3], "k-", linewidth=2)
plt.plot([2.45, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.plot([4.95, 4.95], [0, 1.75], "k:", linewidth=2)
plt.plot([4.85, 4.85], [1.75, 3], "k:", linewidth=2)
plt.text(1.40, 1.0, "Depth=0", fontsize=15)
plt.text(3.2, 1.80, "Depth=1", fontsize=13)
plt.text(4.05, 0.5, "(Depth=2)", fontsize=11)

plt.show()
```



In [4]:

```
### альтернатива https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html

#from graphviz import Source
#from sklearn.tree import export_graphviz

#dot_data = export_graphviz(
#        tree_clf,
#        out_file= None, #"iris_tree.dot",
#        feature_names=iris.feature_names[2:],
#        class_names=iris.target_names,
#        rounded=True,
#        filled=True
#    )

#Source(dot_data)
```

Перенавчання дерев

In [5]:

```
from sklearn.datasets import make_moons
Xm, ym = make_moons(n_samples=100, noise=0.25, random_state=53)
```

In [6]:

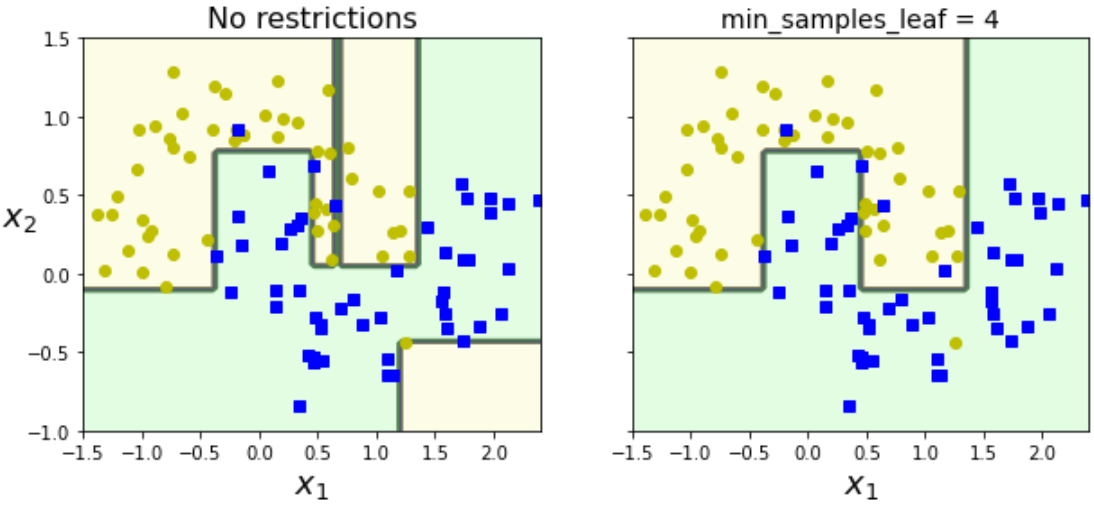
```
deep_tree_clf1 = DecisionTreeClassifier(random_state=42)
deep_tree_clf2 = DecisionTreeClassifier(min_samples_leaf=4, random_state=42)
deep_tree_clf1.fit(Xm, ym)
deep_tree_clf2.fit(Xm, ym)
```

Out[6]:

```
DecisionTreeClassifier(min_samples_leaf=4, random_state=42)
```

In [7]:

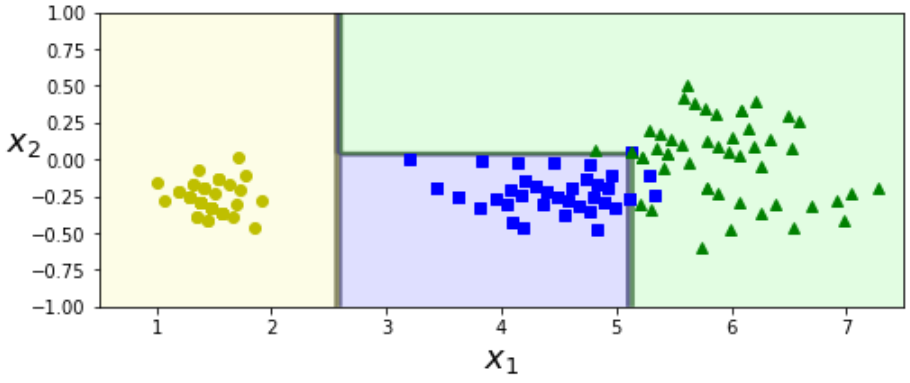
```
fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(deep_tree_clf1, Xm, ym, axes=[-1.5, 2.4, -1, 1.5], iris=False)
plt.title("No restrictions", fontsize=16)
plt.sca(axes[1])
plot_decision_boundary(deep_tree_clf2, Xm, ym, axes=[-1.5, 2.4, -1, 1.5], iris=False)
plt.title("min_samples_leaf = {}".format(deep_tree_clf2.min_samples_leaf), fontsize=14)
plt.ylabel("")

plt.show()
```



Нестійкість до повороту

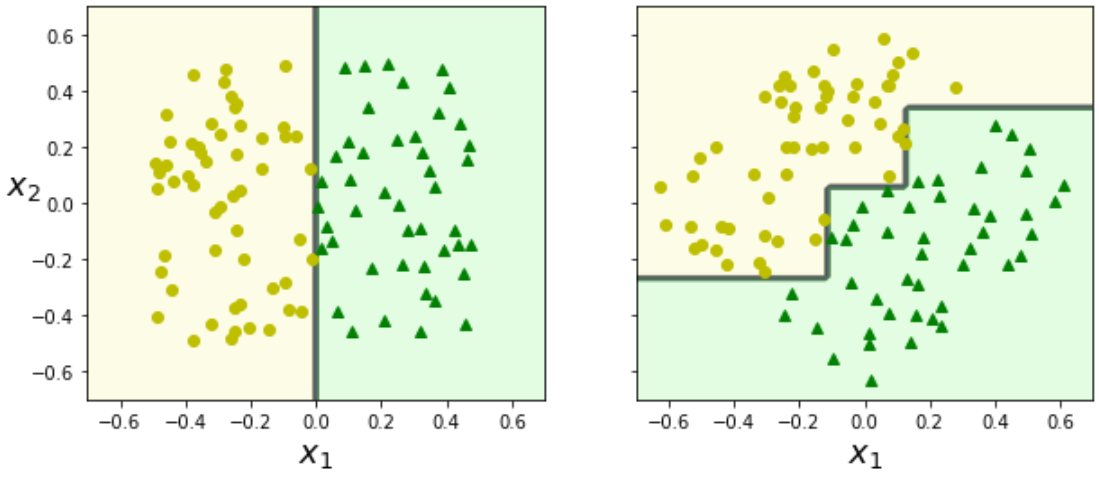
In [8]:

```
angle = np.pi / 180 * 20
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
Xr = X.dot(rotation_matrix)

tree_clf_r = DecisionTreeClassifier(random_state=42)
tree_clf_r.fit(Xr, y)

plt.figure(figsize=(8, 3))
plot_decision_boundary(tree_clf_r, Xr, y, axes=[0.5, 7.5, -1.0, 1], iris=False)

plt.show()
```



In [9]:

```
np.random.seed(6)
Xs = np.random.rand(100, 2) - 0.5
ys = (Xs[:, 0] > 0).astype(np.float32) * 2

angle = np.pi / 4
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
Xsr = Xs.dot(rotation_matrix)

tree_clf_s = DecisionTreeClassifier(random_state=42)
tree_clf_s.fit(Xs, ys)
tree_clf_sr = DecisionTreeClassifier(random_state=42)
tree_clf_sr.fit(Xsr, ys)

fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(tree_clf_s, Xs, ys, axes=[-0.7, 0.7, -0.7, 0.7], iris=False)
plt.sca(axes[1])
plot_decision_boundary(tree_clf_sr, Xsr, ys, axes=[-0.7, 0.7, -0.7, 0.7], iris=False)
plt.ylabel("")

plt.show()
```



Дерева регресії

In [10]:

```
# Quadratic training set + noise
np.random.seed(42)
m = 200
X = np.random.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + np.random.randn(m, 1) / 10
```

In [11]:

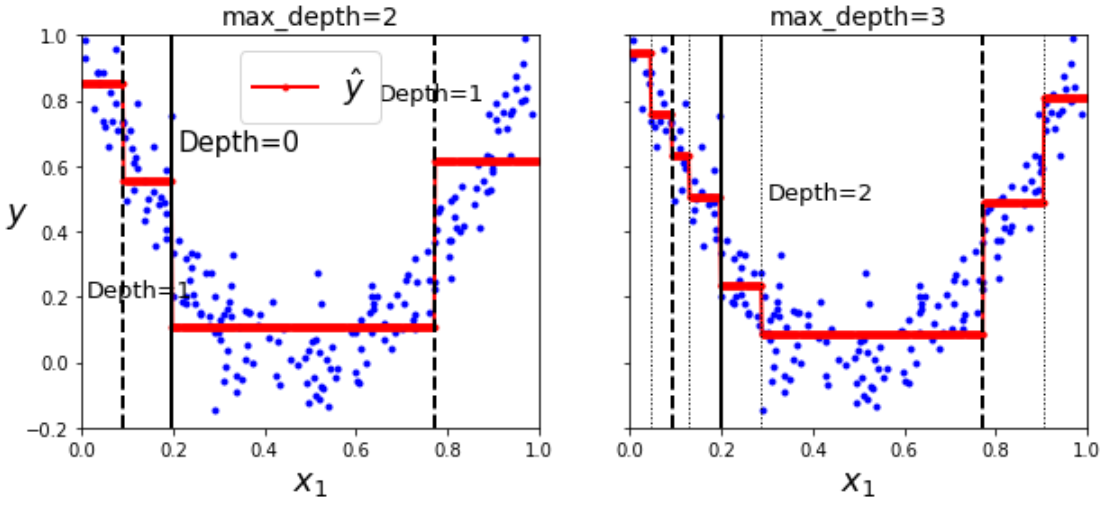
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)
```

Out[11]:

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

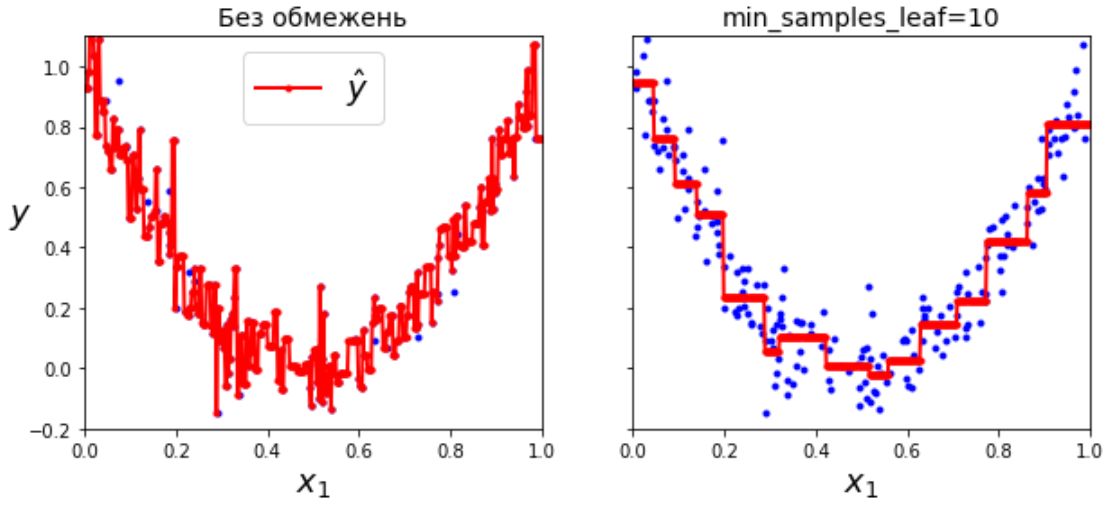
In [12]:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(random_state=42, max_depth=2)
tree_reg2 = DecisionTreeRegressor(random_state=42, max_depth=3)
tree_reg1.fit(X, y)
tree_reg2.fit(X, y)

def plot_regression_predictions(tree_reg, X, y, axes=[0, 1, -0.2, 1], ylabel="$y$"):
    x1 = np.linspace(axes[0], axes[1], 500).reshape(-1, 1)
    y_pred = tree_reg.predict(x1)
    plt.axis(axes)
    plt.xlabel("$x_1$", fontsize=18)
    if ylabel:
        plt.ylabel(ylabel, fontsize=18, rotation=0)
    plt.plot(X, y, "b.")
    plt.plot(x1, y_pred, "r.-", linewidth=2, label=r"$\hat{y}$")

fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)
plt.sca(axes[0])
plot_regression_predictions(tree_reg1, X, y)
for split, style in ((0.1973, "k-"), (0.0917, "k--"), (0.7718, "k--")):
    plt.plot([split, split], [-0.2, 1], style, linewidth=2)
plt.text(0.21, 0.65, "Depth=0", fontsize=15)
plt.text(0.01, 0.2, "Depth=1", fontsize=13)
plt.text(0.65, 0.8, "Depth=1", fontsize=13)
plt.legend(loc="upper center", fontsize=18)
plt.title("max_depth=2", fontsize=14)

plt.sca(axes[1])
plot_regression_predictions(tree_reg2, X, y, ylabel=None)
for split, style in ((0.1973, "k-"), (0.0917, "k--"), (0.7718, "k--")):
    plt.plot([split, split], [-0.2, 1], style, linewidth=2)
for split in (0.0458, 0.1298, 0.2873, 0.9040):
    plt.plot([split, split], [-0.2, 1], "k:", linewidth=1)
plt.text(0.3, 0.5, "Depth=2", fontsize=13)
plt.title("max_depth=3", fontsize=14)

plt.show()
```


max_depth=2
max_depth=3
$\hat{y}$
Depth=0
Depth=1
Depth=1
Depth=2
y
x_1
x_1

In [13]:

```
tree_reg1 = DecisionTreeRegressor(random_state=42)
tree_reg2 = DecisionTreeRegressor(random_state=42, min_samples_leaf=10)
tree_reg1.fit(X, y)
tree_reg2.fit(X, y)

x1 = np.linspace(0, 1, 500).reshape(-1, 1)
y_pred1 = tree_reg1.predict(x1)
y_pred2 = tree_reg2.predict(x1)

fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)

plt.sca(axes[0])
plt.plot(X, y, "b.")
plt.plot(x1, y_pred1, "r.-", linewidth=2, label=r"$\hat{y}$")
plt.axis([0, 1, -0.2, 1.1])
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", fontsize=18, rotation=0)
plt.legend(loc="upper center", fontsize=18)
plt.title("Без обмежень", fontsize=14)

plt.sca(axes[1])
plt.plot(X, y, "b.")
plt.plot(x1, y_pred2, "r.-", linewidth=2, label=r"$\hat{y}$")
plt.axis([0, 1, -0.2, 1.1])
plt.xlabel("$x_1$", fontsize=18)
plt.title("min_samples_leaf={}".format(tree_reg2.min_samples_leaf), fontsize=14)

plt.show()
```



Bagging ensembles

In [14]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [15]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [16]:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.904
```

In [17]:

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

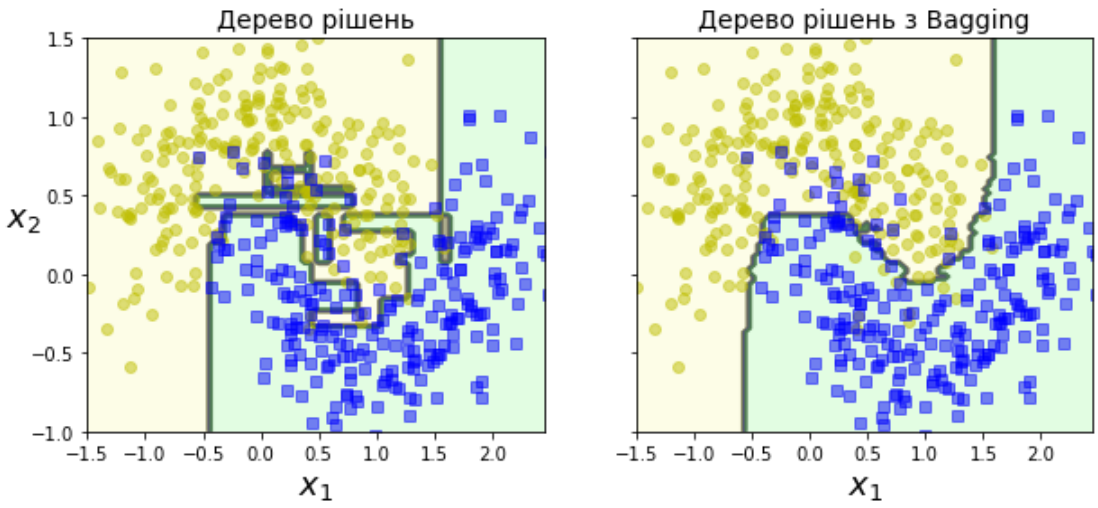
```
0.856
```

In [18]:

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
```

In [19]:

```
fix, axes = plt.subplots(ncols=2, figsize=(10,4), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(tree_clf, X, y)
plt.title("Дерево рішень", fontsize=14)
plt.sca(axes[1])
plot_decision_boundary(bag_clf, X, y)
plt.title("Дерево рішень з Bagging", fontsize=14)
plt.ylabel("")
plt.show()
```



Random Forests

In [20]:

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),
    n_estimators=500, max_samples=1.0, bootstrap=True, random_state=42)
```

In [21]:

```
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [22]:

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

In [23]:

```
np.sum(y_pred == y_pred_rf) / len(y_pred)  # майже однакові виходи моделей
```

Out[23]:

```
0.976
```

In [24]:

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
```

In [25]:

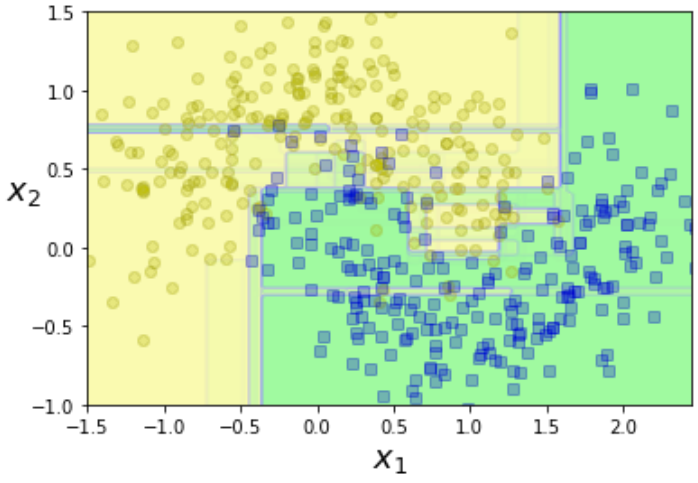
```
rnd_clf.feature_importances_
```

Out[25]:

```
array([0.11249225, 0.02311929, 0.44103046, 0.423358  ])
```

In [26]:

```
plt.figure(figsize=(6, 4))

for i in range(15):
    tree_clf = DecisionTreeClassifier(max_leaf_nodes=16, random_state=42 + i)
    indices_with_replacement = np.random.randint(0, len(X_train), len(X_train))
    tree_clf.fit(X[indices_with_replacement], y[indices_with_replacement])
    plot_decision_boundary(tree_clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.02, contour=False)

plt.show()
```



Ансамбль методом голосування

In [27]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [28]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", random_state=42)
```

Жорстке голосування:

In [29]:

```
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
```

In [30]:

```
voting_clf.fit(X_train, y_train)
```

Out[30]:

```
VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                             ('rf', RandomForestClassifier(random_state=42)),
                             ('svc', SVC(random_state=42))])
```

In [31]:

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.912
```

М'яке голосування:

In [32]:

```
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", probability=True, random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft')
voting_clf.fit(X_train, y_train)
```

Out[32]:

```
VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                             ('rf', RandomForestClassifier(random_state=42)),
                             ('svc', SVC(probability=True, random_state=42))],
                 voting='soft')
```

In [33]:

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.92
```

Завдання

Вправа 1: натренуйте модель дерева рішень та виконайте пошук гіперпараметрів для moons dataset. \ a. Згенеруйте датасет, використовуючи функцію `make_moons(n_samples=10000, noise=0.4)` . \ b. Розбийте її на тренувальну та тестувальну частини, використовуючи функцію `train_test_split()` . \ c. Використайте пошук з крос-валідацією (`GridSearchCV` , `RandomizedSearchCV`) для пошуку гіперпараметрів моделі `DecisionTreeClassifier` . Зокрема, спробуйте різні значення для параметра `max_leaf_nodes` .

In [34]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
np.random.seed(42)

X,y = make_moons(n_samples=10000,noise=0.4)
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

In [35]:

```
clf = DecisionTreeClassifier()
params = {'max_leaf_nodes':[3,5,10,15,20,25,30,50,100,None],
          'max_depth':[5,25,50,100,None],
          'min_samples_leaf':[1,2,3,.1,.2,.3],
          'criterion':['gini','entropy']
         }
grid_search = GridSearchCV(clf, params,n_jobs=-1,cv=3,verbose=0)
grid_search.fit(X_train,y_train)

best_estimator = grid_search.best_estimator_
print(f"clf: {best_estimator}\ntrain_acc: {accuracy_score(y_train,best_estimator.predict(X_train)):.4
f}\ntest_acc: {accuracy_score(y_test,best_estimator.predict(X_test)):.4f}")
```

```
clf: DecisionTreeClassifier(max_depth=25, max_leaf_nodes=20)
train_acc: 0.8705
test_acc: 0.8600
```

In [36]:

```
distributions = {'max_leaf_nodes':[3,5,10,15,20,25,30,50,100,None],
                 'max_depth': [5,25,50,100, None],
                 'max_features': [1,2,None],
                 'min_samples_leaf': np.concatenate([np.linspace(2,4,num=3,dtype=np.int32),np.linspac
e(0,0.4,num=5)]),
                 'criterion': ["gini", "entropy"]}
random_search = RandomizedSearchCV(clf,distributions,n_jobs=-1,cv=3,n_iter=1000,verbose=2)
%time random_search.fit(X_train,y_train)

best_estimator = random_search.best_estimator_
print(f"clf: {best_estimator}\ntrain_acc: {accuracy_score(y_train,best_estimator.predict(X_train)):.4
f}\ntest_acc: {accuracy_score(y_test,best_estimator.predict(X_test)):.4f}")
```

```
Fitting 3 folds for each of 1000 candidates, totalling 3000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:    0.1s

Wall time: 2.61 s
clf: DecisionTreeClassifier(criterion='entropy', max_features=1, max_leaf_nodes=5,
                       min_samples_leaf=0.2)
train_acc: 0.8554
test_acc: 0.8390

[Parallel(n_jobs=-1)]: Done 3000 out of 3000 | elapsed:    2.5s finished
```

Вправа 2 : \ a. Завантажте датасет MNIST та розбийте його на тренувальну, валідаційну та тестувальну частини (наприклад, 50 000 / 10 000 / 10 000). \ b. Натренуйте різні класифікаційні моделі (Random Forest classifier, Logistic Regression, SVM). \ c. Далі, об'єднайте їх за допомогою голосування. \ d. Натренуйте нову модель, використовуючи виходи попередніх моделей на валідаційній вибірці (це будуть нові ознаки і нова тренувальна вибірка для даної моделі). Протестуйте отриманий ланцюжок на тренувальний вибірці. Такий спосіб поєднання моделей називається стогуванням (stacking).

In [37]:

```
from sklearn.svm import LinearSVC
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1, cache=True)

X = mnist["data"]
y = mnist["target"].astype(np.uint8)

X_train_full = X[:60000]
y_train_full = y[:60000]
X_test = X[60000:]
y_test = y[60000:]

X_train,X_val,y_train,y_val = train_test_split(X_train_full,y_train_full,test_size=10000)
```

Using different classifiers and classification via voting

I'll use tuned LinearSVC instead of SVC (converges faster)

In [38]:

```
%%time
rf_clf = RandomForestClassifier()
lr_clf = LogisticRegression()
svc_clf = LinearSVC(max_iter=1000,n_jobs-1)

voting_clf = VotingClassifier(
    n_jobs=-1,
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft')

for clf in [rf_clf, lr_clf, svc_clf, voting_clf]:
    %time clf.fit(X_train,y_train)
    print(f"clf:{clf.__class__.__name__}\ntrain accuracy:{accuracy_score(y_train,clf.predict(X_train
)):.4f}\ntest accuracy: {accuracy_score(y_test,clf.predict(X_test)):.4f}\n")
```

```
Wall time: 38.3 s
clf:RandomForestClassifier
train accuracy:1.0000
test accuracy: 0.9686


C:\Users\eddie\kma_course_env\lib\site-packages\sklearn\linear_model\_logistic.py:762: C
onvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Wall time: 16.9 s
clf:LogisticRegression
train accuracy:0.9345
test accuracy: 0.9236


C:\Users\eddie\kma_course_env\lib\site-packages\sklearn\svm\_base.py:976: ConvergenceWar
ning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "

Wall time: 1min 59s
clf:LinearSVC
train accuracy:0.8756
test accuracy: 0.8673

Wall time: 33min 16s
clf:VotingClassifier
train accuracy:0.9910
test accuracy: 0.9701

Wall time: 47min 45s
```

Stacking

Using classifier predictions as only features

In [39]:

```
data,new_data = [X_train,X_test,X_val],[None,None,None]

for clf in [rf_clf,lr_clf,svc_clf]:
    for j in range(3):
        pred = clf.predict(data[j]).reshape(-1,1)
        if new_data[j] is None:
            new_data[j] = pred
        else:
            new_data[j] = np.concatenate((new_data[j],pred),axis=1)

new_X_train,new_X_test,new_X_val = new_data

new_model = DecisionTreeClassifier()
new_model.fit(new_X_val,y_val)
print(f"clf:{new_model.__class__.__name__}\ntrain accuracy:{accuracy_score(y_train,new_model.predict(
new_X_train))}\ntest accuracy: {accuracy_score(y_test,new_model.predict(new_X_test)):.4f}\n")
```

```
clf:DecisionTreeClassifier
train accuracy:0.9934
test accuracy: 0.9658
```