Decision Trees for the Iris Dataset

Run in Google Colab (https://colab.research.google.com/github/ageron/handson-ml2/blob/master/06_decision_trees.ipynb)

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20.

In [1]:

```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resoluti
on=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Training and visualizing

In [59]:

```
from sklearn.datasets import load_iris # load the dataset
from sklearn.tree import DecisionTreeClassifier # we use this library
from Scikit Learn

iris = load_iris()
X = iris.data[:, 2:] # the attributes of the data item. Note: petal le
ngth and width only, don't use the 3rd attribute
y = iris.target  # the labels of the data item

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)  # de
cision tree depth should be at most 2
tree_clf.fit(X, y)
```

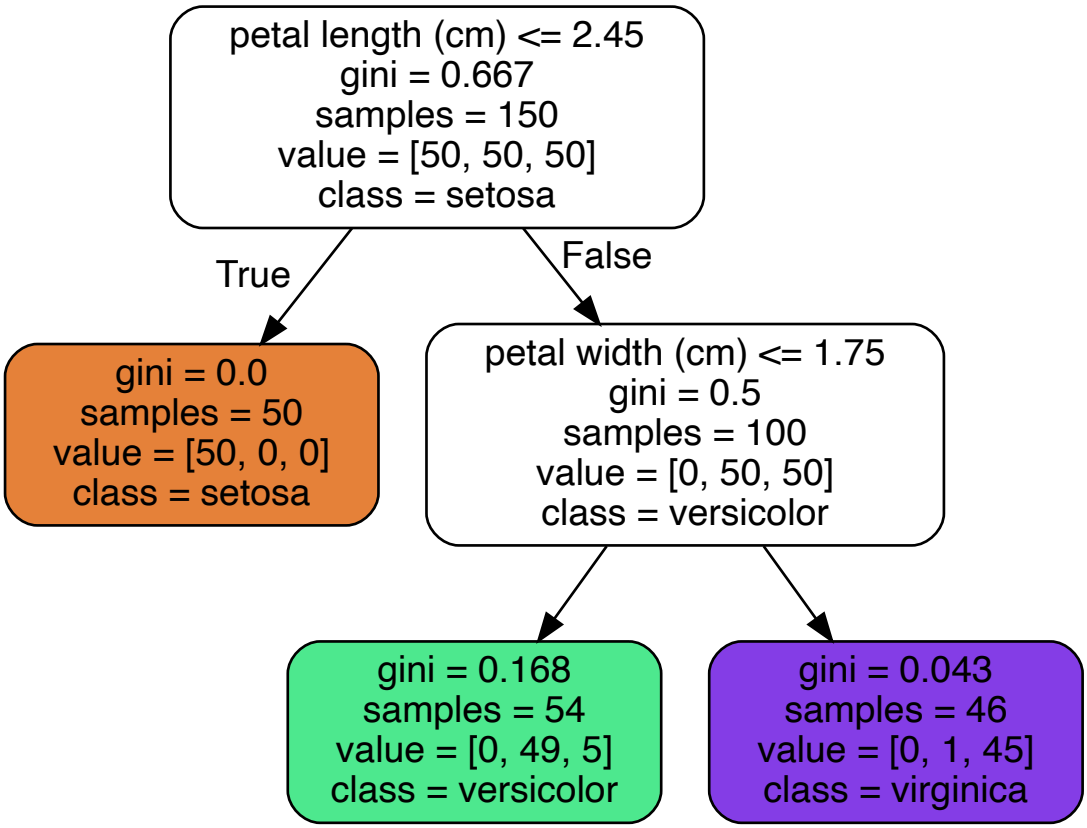
Out[59]:

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

In [60]:

```
# for visualizing the decision tree
from graphviz import Source
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

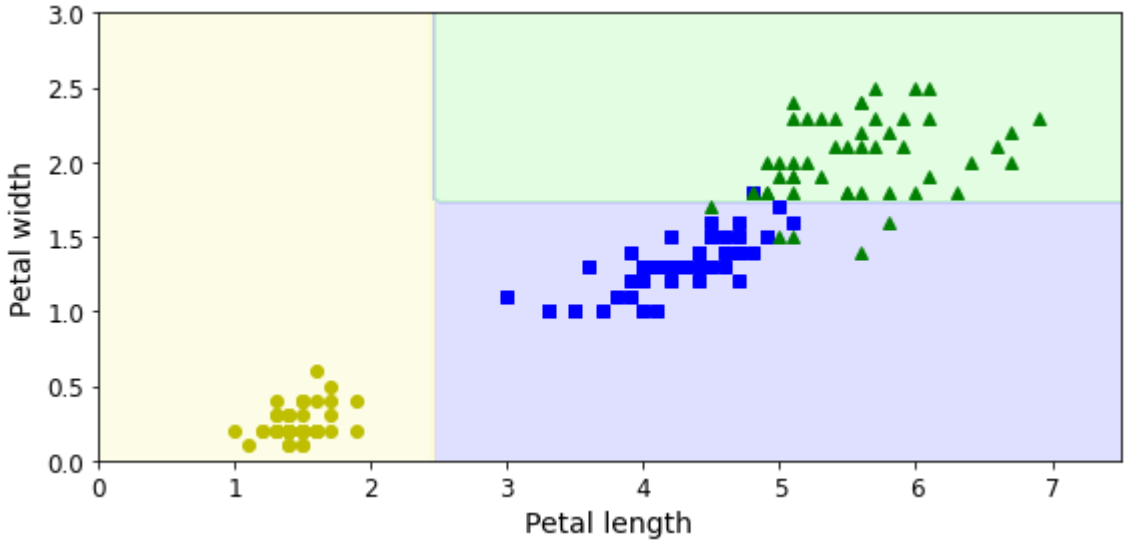
Out[60]:



In [81]:

```
# Plot the decision tree as a partion of the attribute-values space

from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True,
legend=False, plot_training=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 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 seto
sa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris vers
icolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris virg
inica")
        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)

save_fig("decision_tree_decision_boundaries_plot")
plt.show()
```

```
Saving figure decision_tree_decision_boundaries_plot
```



Predicting classes and class probabilities

In [5]:
```
tree_clf.predict_proba([[5, 1.5]])  # probabilities of each class
```

Out[5]:
```
array([[0.        , 0.90740741, 0.09259259]])
```

In [6]:
```
tree_clf.predict([[5, 1.5]]) # outputs the class with the highest pro
bability
```

Out[6]:
```
array([1])
```

Sensitivity to training set details

In [76]:
```
X[(X[:, 0]==X[:, 0][y==1].max()) & (y==1)] # longest Iris versicolor
flower
```

Out[76]:
```
array([[5.1, 1.6]])
```

In [77]:

```
X[(X[:, 0]==X[:, 0][y==2].min()) & (y==2)] # shortest Iris virginica flower
```

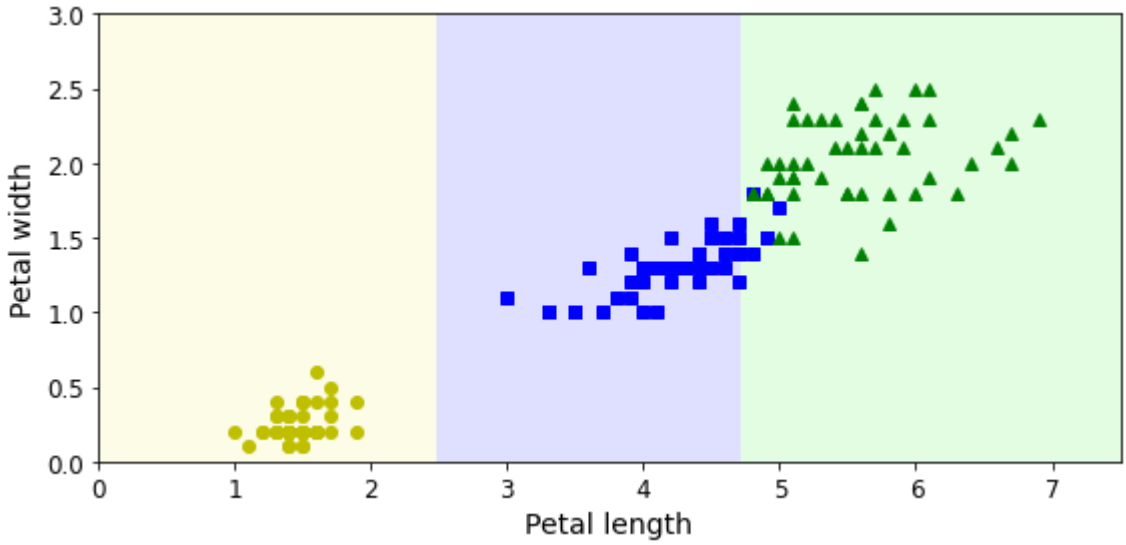
Out[77]:

```
array([[4.5, 1.7]])
```

In [79]:

```
eliminate_some= ((X[:, 0]<=5.0) & (y==1)) | ((X[:, 0]>=4.6) & (y==2)) | (y==0)  # remove longest versicolor and shortest virginica
X_tweaked = X[eliminate_some]
y_tweaked = y[eliminate_some]

tree_clf_tweaked = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_tweaked.fit(X_tweaked, y_tweaked)

plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_tweaked, X_tweaked, y_tweaked, legend=False)


save_fig("decision_tree_instability_plot")
plt.show()
```

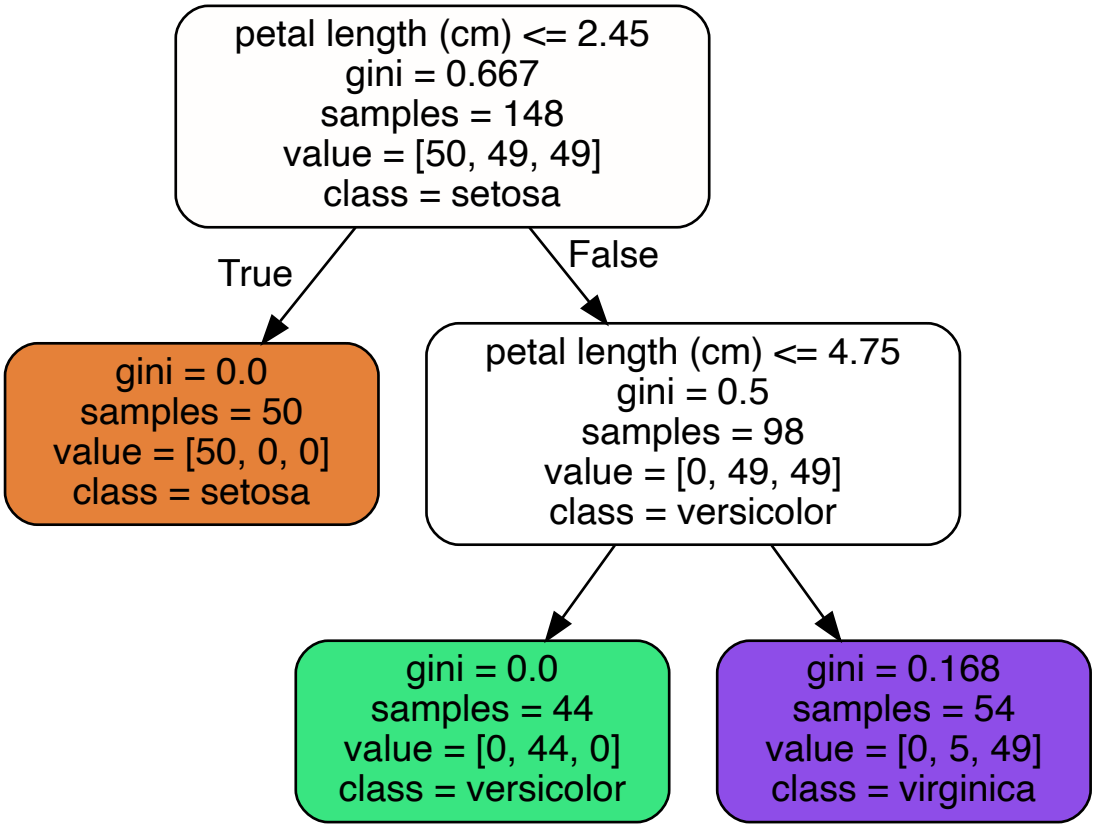
```
Saving figure decision_tree_instability_plot
```



In [82]:

```
export_graphviz(
        tree_clf_tweaked,
        out_file=os.path.join(IMAGES_PATH, "iris_tree_tweaked.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree_tweaked.dot"))
```

Out[82]:

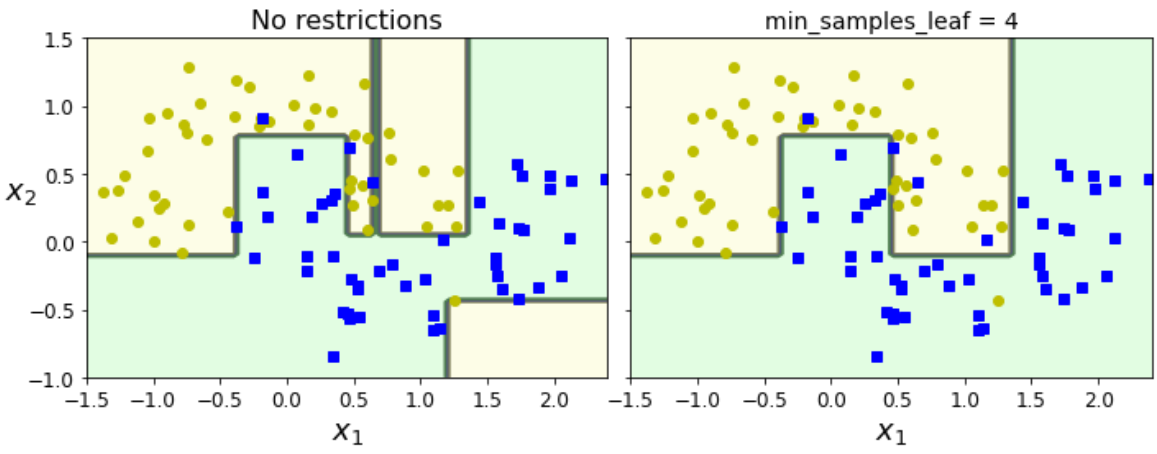


On an artificially generated dataset

In [11]:

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

deep_tree_clf1 = DecisionTreeClassifier(random_state=42)
deep_tree_clf2 = DecisionTreeClassifier(min_samples_leaf=4, random_st
ate=42) # split a node further only if it has at least 4 data points
deep_tree_clf1.fit(Xm, ym)
deep_tree_clf2.fit(Xm, ym)

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_l
eaf), fontsize=14)
plt.ylabel("")

save_fig("min_samples_leaf_plot")
plt.show()
```

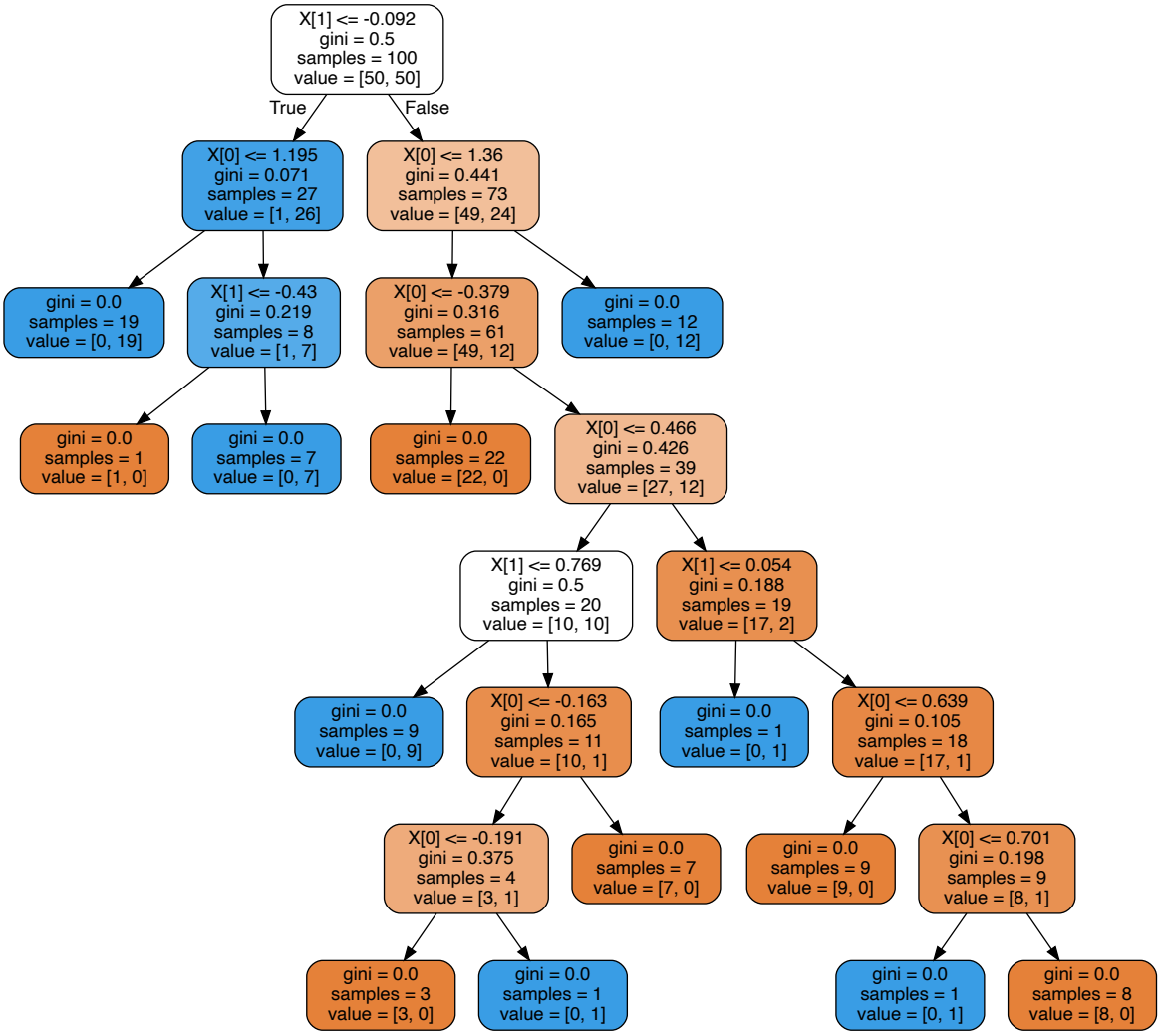
```
Saving figure min_samples_leaf_plot
```



In [12]:

```
export_graphviz(
        deep_tree_clf1,
        out_file=os.path.join(IMAGES_PATH, "deep_tree_clf1.dot"),
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "deep_tree_clf1.dot"))
```

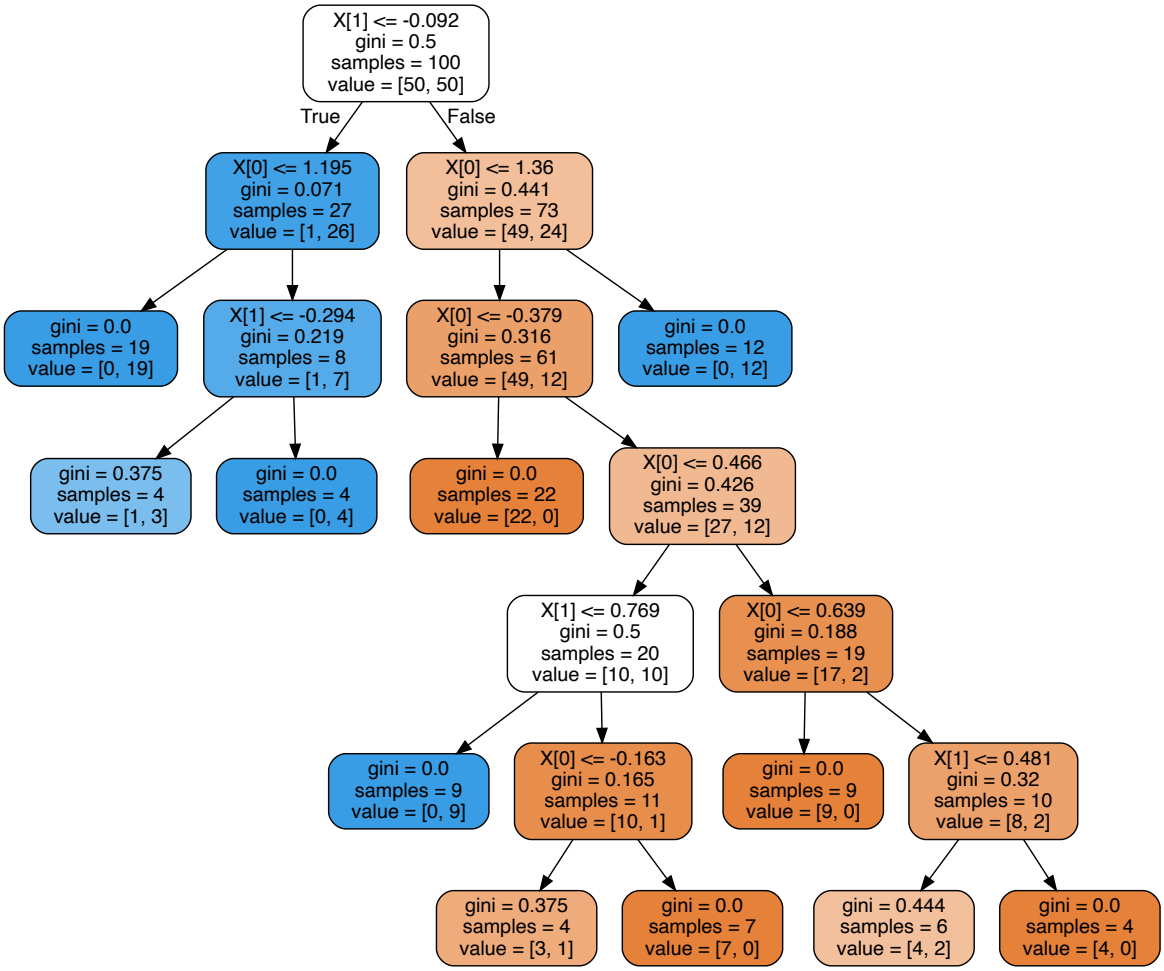
Out[12]:



In [13]:

```
export_graphviz(
        deep_tree_clf2,
        out_file=os.path.join(IMAGES_PATH, "deep_tree_clf2.dot"),
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "deep_tree_clf2.dot"))
```

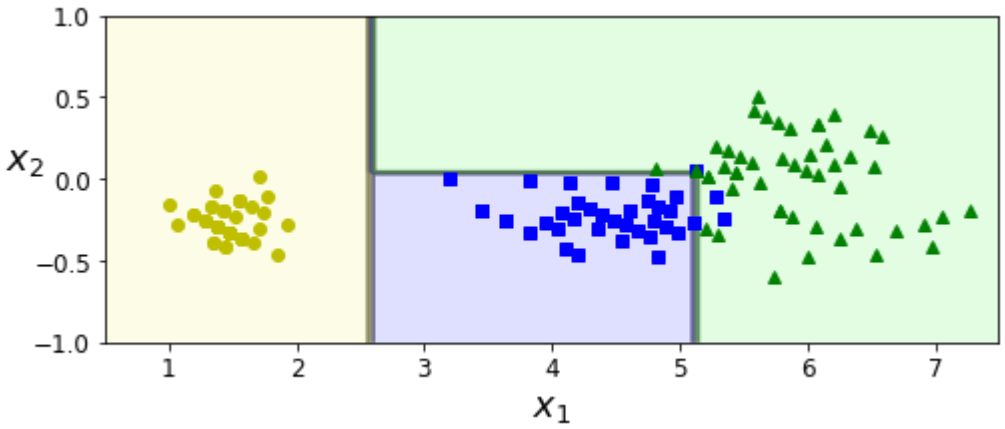
Out[13]:



Effect of rotating the data points in the plane

In [14]:

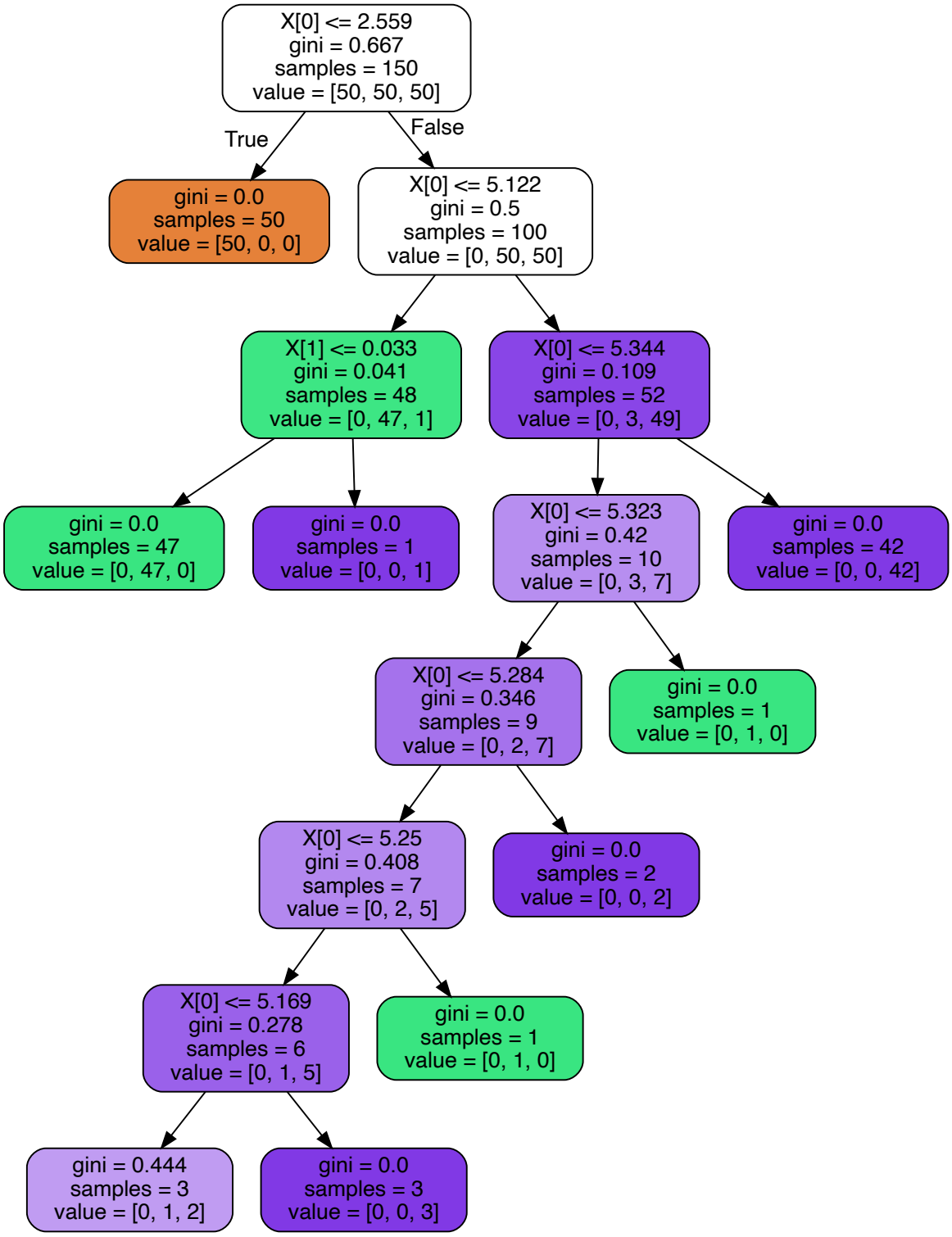
```
angle = np.pi / 180 * 20
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin
(angle), np.cos(angle)]])
Xr = X.dot(rotation_matrix)

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], i
ris=False)

plt.show()
```



In [15]:

```
export_graphviz(
        tree_clf_r,
        out_file=os.path.join(IMAGES_PATH, "tree_clf_r.dot"),
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "tree_clf_r.dot"))
```

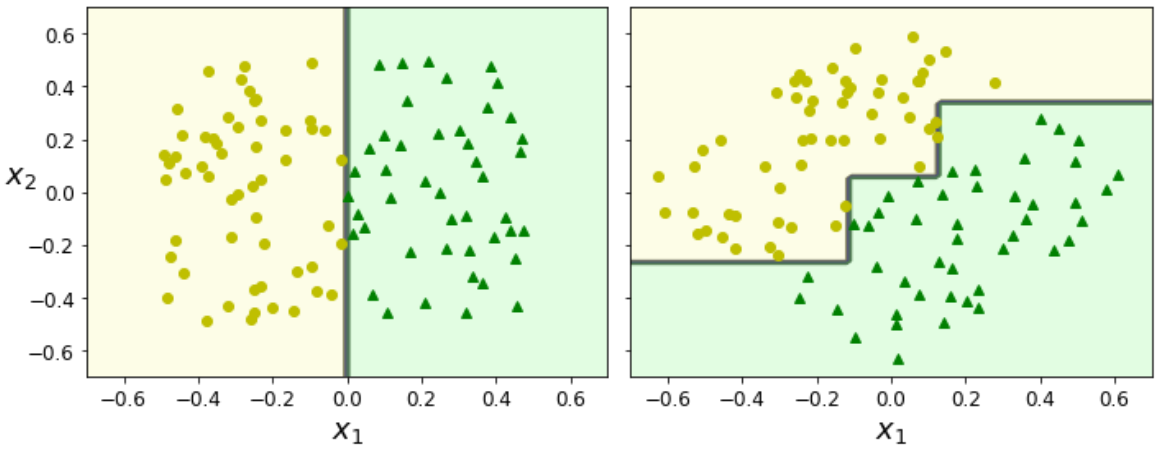
Out[15]:


X[0] <= 2.559
gini = 0.667
samples = 150
value = [50, 50, 50]
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
X[0] <= 5.122
gini = 0.5
samples = 100
value = [0, 50, 50]
X[1] <= 0.033
gini = 0.041
samples = 48
value = [0, 47, 1]
X[0] <= 5.344
gini = 0.109
samples = 52
value = [0, 3, 49]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
X[0] <= 5.323
gini = 0.42
samples = 10
value = [0, 3, 7]
gini = 0.0
samples = 42
value = [0, 0, 42]
X[0] <= 5.284
gini = 0.346
samples = 9
value = [0, 2, 7]
gini = 0.0
samples = 1
value = [0, 1, 0]
X[0] <= 5.25
gini = 0.408
samples = 7
value = [0, 2, 5]
gini = 0.0
samples = 2
value = [0, 0, 2]
X[0] <= 5.169
gini = 0.278
samples = 6
value = [0, 1, 5]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 3
value = [0, 0, 3]

In [16]:

```
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("")

save_fig("sensitivity_to_rotation_plot")
plt.show()
```

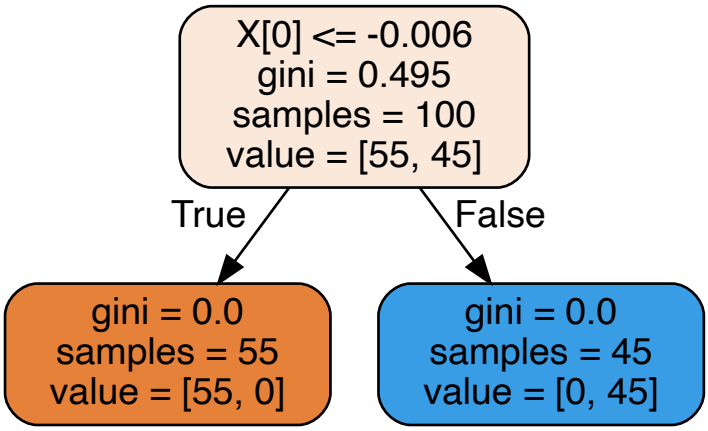
```
Saving figure sensitivity_to_rotation_plot
```



In [17]:

```
export_graphviz(
        tree_clf_s,
        out_file=os.path.join(IMAGES_PATH, "tree_clf_s.dot"),
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "tree_clf_s.dot"))
```

Out[17]:



In [18]:

```
export_graphviz(
        tree_clf_sr,
        out_file=os.path.join(IMAGES_PATH, "tree_clf_sr.dot"),
        rounded=True,
        filled=True
    )

Source.from_file(os.path.join(IMAGES_PATH, "tree_clf_sr.dot"))
```

Out[18]: