3주차 실습과제

2016314786 김호진

[6장 실습 – 의사결정나무모델]

jupyter 6장 의사결정나무모델 Last Checkpoint: 5분 전 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | machinelearning

Run Code

In [1]:
```
import warnings
warnings.filterwarnings(action='ignore')
```

In [2]:
```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sn
```

In [3]:
```
iris = load_iris()
iris_frame = pd.DataFrame(data=np.c_[iris['data'], iris['target']],columns = iris['feature_names'] + ['target'])
iris_frame['target'] = iris_frame['target'].map({1:"versicolor",0:"setosa",2:"virginica"})
X = iris_frame.iloc[:,:-1]
Y = iris_frame.iloc[:,[-1]]
iris_frame
```

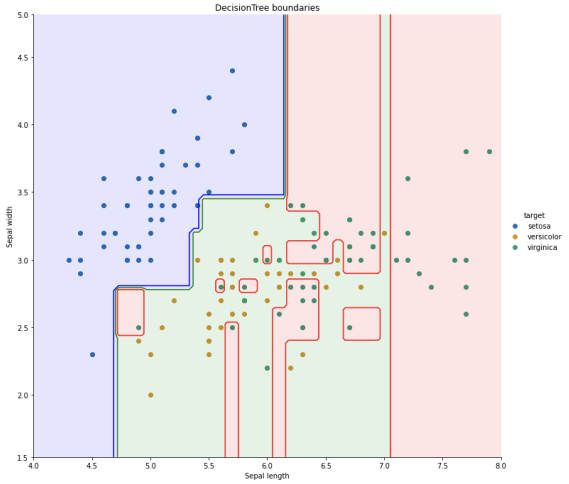
Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

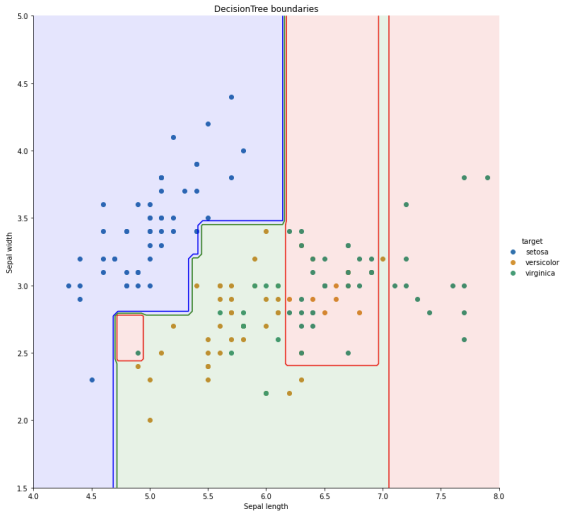
In [4]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0, criterion='gini', max_depth=10)
import matplotlib.colors as colors

df1 = iris_frame[["sepal length (cm)" , "sepal width (cm)", "target" ]]
X = df1.iloc[:,0:2]
Y = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

clf.fit(X,Y)
N=100
```

In [5]:
```
X_= np.linspace(4,8,N)
Y_= np.linspace(1.5,5,N)
X_,Y_= np.meshgrid(X_,Y_)

color_list= ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1,vmax=1)
g= sn.FacetGrid(iris_frame, hue = "target", size =10,
                palette = 'colorblind').map(plt.scatter, "sepal length (cm)","sepal width (cm)",).add_legend()

my_ax = g.ax
ZZ = np.array( [clf.predict ([[xx,yy]]) [0] for xx,yy in zip(np.ravel(X_),np.ravel(Y_))])
Z=zz.reshape(X_.shape)

my_ax.contourf(X_,Y_,Z,2,alpha=.1, colors = ('blue','green','red'))
my_ax.contour(X_,Y_,Z,2,alpha=1, colors = ('blue','green','red'))

my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('DecisionTree boundaries')

plt.show()
```



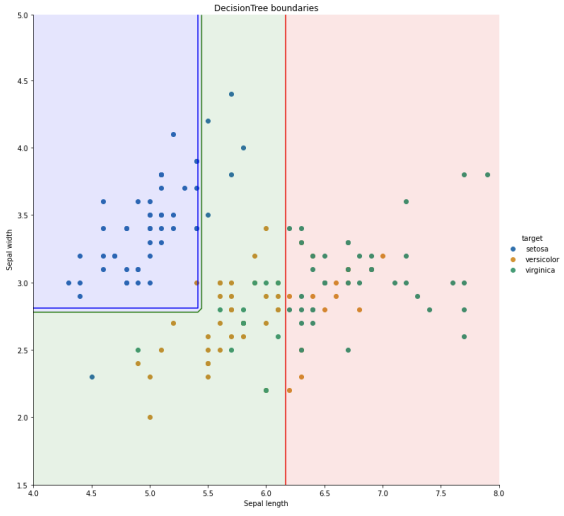
In [6]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0, criterion='gini', max_depth=5)
import matplotlib.colors as colors

df1 = iris_frame[["sepal length (cm)" , "sepal width (cm)", "target" ]]
X = df1.iloc[:,0:2]
Y = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

clf.fit(X,Y)
N=100
```

In [7]:
```
X_= np.linspace(4,8,N)
Y_= np.linspace(1.5,5,N)
X_,Y_= np.meshgrid(X_,Y_)

color_list= ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1,vmax=1)
g= sn.FacetGrid(iris_frame, hue = "target", size =10,
                palette = 'colorblind').map(plt.scatter, "sepal length (cm)","sepal width (cm)",).add_legend()

my_ax = g.ax
zz = np.array( [clf.predict ([[xx,yy]]) [0] for xx,yy in zip(np.ravel(X_),np.ravel(Y_))])
Z=zz.reshape(X_.shape)

my_ax.contourf(X_,Y_,Z,2,alpha=.1, colors = ('blue','green','red'))
my_ax.contour(X_,Y_,Z,2,alpha=1, colors = ('blue','green','red'))

my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('DecisionTree boundaries')

plt.show()
```



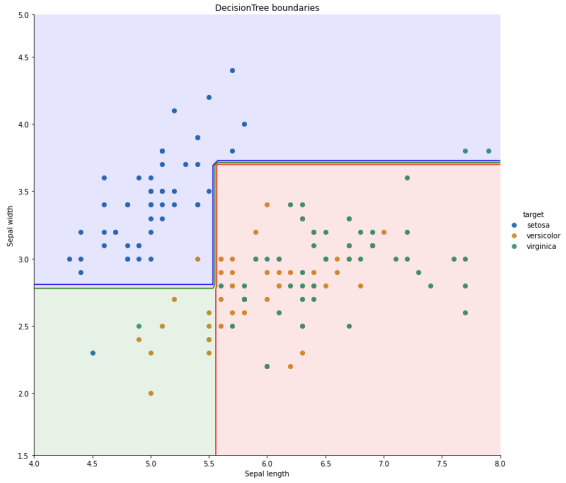
In [8]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0, criterion='gini', max_depth=2)
import matplotlib.colors as colors

df1 = iris_frame[["sepal length (cm)" , "sepal width (cm)", "target" ]]
X = df1.iloc[:,0:2]
Y = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

clf.fit(X,Y)
N=100
```

In [9]:
```
X_= np.linspace(4,8,N)
Y_= np.linspace(1.5,5,N)
X_,Y_= np.meshgrid(X_,Y_)

color_list= ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1,vmax=1)
g= sn.FacetGrid(iris_frame, hue = "target", size =10,
                palette = 'colorblind').map(plt.scatter, "sepal length (cm)","sepal width (cm)",).add_legend()

my_ax = g.ax
zz = np.array( [clf.predict ([[xx,yy]]) [0] for xx,yy in zip(np.ravel(X_),np.ravel(Y_))])
Z=zz.reshape(X_.shape)

my_ax.contourf(X_,Y_,Z,2,alpha=.1, colors = ('blue','green','red'))
my_ax.contour(X_,Y_,Z,2,alpha=1, colors = ('blue','green','red'))

my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('DecisionTree boundaries')

plt.show()
```



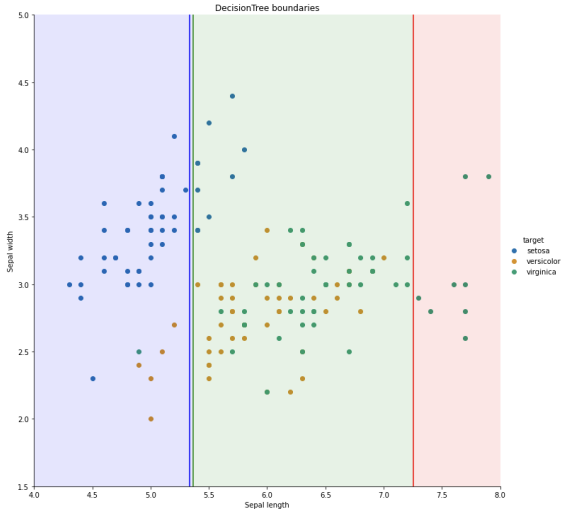
In [10]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0, criterion='entropy', max_depth=2)
import matplotlib.colors as colors

df1 = iris_frame[["sepal length (cm)" , "sepal width (cm)", "target" ]]
X = df1.iloc[:,0:2]
Y = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

clf.fit(X,Y)
N=100
```

In [11]:
```
X_= np.linspace(4,8,N)
Y_= np.linspace(1.5,5,N)
X_,Y_= np.meshgrid(X_,Y_)

color_list= ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1,vmax=1)
g= sn.FacetGrid(iris_frame, hue = "target", size =10,
                palette = 'colorblind').map(plt.scatter, "sepal length (cm)","sepal width (cm)",).add_legend()

my_ax = g.ax
zz = np.array( [clf.predict ([[xx,yy]]) [0] for xx,yy in zip(np.ravel(X_),np.ravel(Y_))])
Z=zz.reshape(X_.shape)

my_ax.contourf(X_,Y_,Z,2,alpha=.1, colors = ('blue','green','red'))
my_ax.contour(X_,Y_,Z,2,alpha=1, colors = ('blue','green','red'))

my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('DecisionTree boundaries')

plt.show()
```

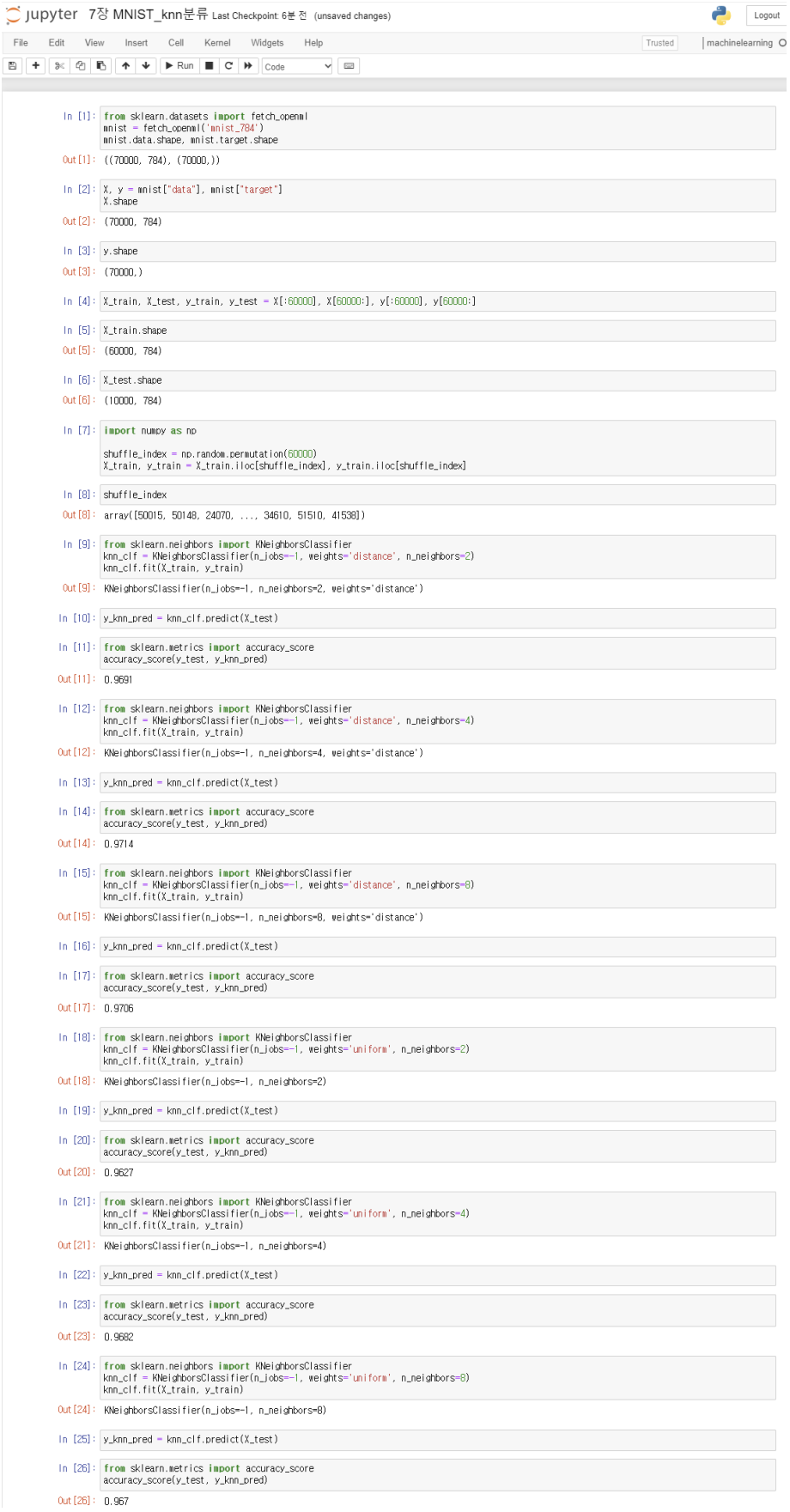


In [12]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0, criterion='gini', max_depth=2, splitter='random')
import matplotlib.colors as colors

df1 = iris_frame[["sepal length (cm)" , "sepal width (cm)", "target" ]]
X = df1.iloc[:,0:2]
Y = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).copy()

clf.fit(X,Y)
N=100
```

In [13]:
```
X_= np.linspace(4,8,N)
Y_= np.linspace(1.5,5,N)
X_,Y_= np.meshgrid(X_,Y_)

color_list= ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1,vmax=1)
g= sn.FacetGrid(iris_frame, hue = "target", size =10,
                palette = 'colorblind').map(plt.scatter, "sepal length (cm)","sepal width (cm)",).add_legend()

my_ax = g.ax
zz = np.array( [clf.predict ([[xx,yy]]) [0] for xx,yy in zip(np.ravel(X_),np.ravel(Y_))])
Z=zz.reshape(X_.shape)

my_ax.contourf(X_,Y_,Z,2,alpha=.1, colors = ('blue','green','red'))
my_ax.contour(X_,Y_,Z,2,alpha=1, colors = ('blue','green','red'))

my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('DecisionTree boundaries')

plt.show()
```



[7장 실습 – MNIST knn분류]



jupyter 7장 MNIST_knn분류 Last Checkpoint: 6분 전 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | machinelearning

Run Code

In [1]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
mnist.data.shape, mnist.target.shape
```
Out[1]: ((70000, 784), (70000,))

In [2]:
```
X, y = mnist["data"], mnist["target"]
X.shape
```
Out[2]: (70000, 784)

In [3]:
```
y.shape
```
Out[3]: (70000,)

In [4]:
```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

In [5]:
```
X_train.shape
```
Out[5]: (60000, 784)

In [6]:
```
X_test.shape
```
Out[6]: (10000, 784)

In [7]:
```
import numpy as np

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train.iloc[shuffle_index], y_train.iloc[shuffle_index]
```

In [8]:
```
shuffle_index
```
Out[8]: array([50015, 50148, 24070, ..., 34610, 51510, 41538])

In [9]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=2)
knn_clf.fit(X_train, y_train)
```
Out[9]: KNeighborsClassifier(n_jobs=-1, n_neighbors=2, weights='distance')

In [10]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [11]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[11]: 0.9691

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=4)
knn_clf.fit(X_train, y_train)
```
Out[12]: KNeighborsClassifier(n_jobs=-1, n_neighbors=4, weights='distance')

In [13]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [14]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[14]: 0.9714

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=8)
knn_clf.fit(X_train, y_train)
```
Out[15]: KNeighborsClassifier(n_jobs=-1, n_neighbors=8, weights='distance')

In [16]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [17]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[17]: 0.9705

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='uniform', n_neighbors=2)
knn_clf.fit(X_train, y_train)
```
Out[18]: KNeighborsClassifier(n_jobs=-1, n_neighbors=2)

In [19]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [20]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[20]: 0.9627

In [21]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='uniform', n_neighbors=4)
knn_clf.fit(X_train, y_train)
```
Out[21]: KNeighborsClassifier(n_jobs=-1, n_neighbors=4)

In [22]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [23]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[23]: 0.9682

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='uniform', n_neighbors=8)
knn_clf.fit(X_train, y_train)
```
Out[24]: KNeighborsClassifier(n_jobs=-1, n_neighbors=8)

In [25]:
```
y_knn_pred = knn_clf.predict(X_test)
```

In [26]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_knn_pred)
```
Out[26]: 0.967

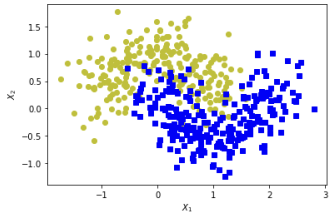
[7장 실습 – make moons, Iris ensemble]

jupyter 7장 Make moons_Iris Ensemble Last Checkpoint: 9분 전 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted machinelearning

Run Code

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

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.show()
```



In [2]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [3]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=100,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [4]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.92
```

In [5]:
```
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, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [6]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

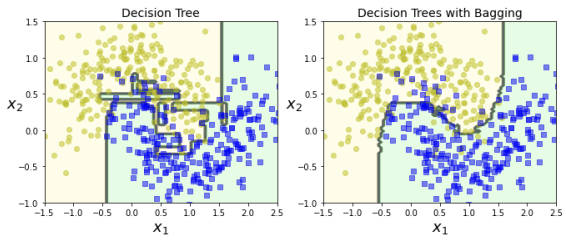
```
0.904
```

In [7]:
```
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 [8]:
```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.5, -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)
```

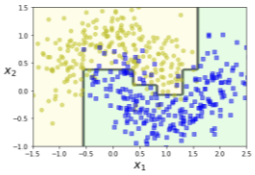
In [9]:
```
plt.figure(figsize=(11,4))
plt.subplot(121)
plot_decision_boundary(tree_clf, X, y)
plt.title("Decision Tree", fontsize=14)
plt.subplot(122)
plot_decision_boundary(bag_clf, X, y)
plt.title("Decision Trees with Bagging", fontsize=14)
plt.show()
```



In [10]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=20,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[10]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=20, random_state=42)
```

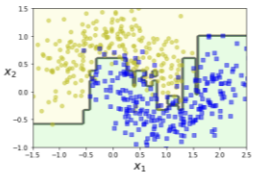
In [11]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [12]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[12]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=200, random_state=42)
```

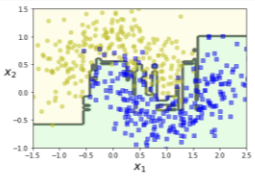
In [13]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [14]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=500,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[14]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=500, random_state=42)
```

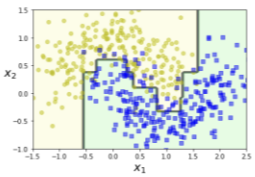
In [15]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [16]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.1, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[16]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.1, n_estimators=200, random_state=42)
```

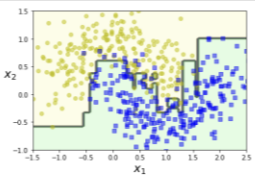
In [17]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [18]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[18]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=200, random_state=42)
```

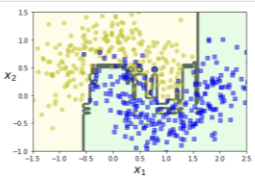
In [19]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [20]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.9, random_state=42)
ada_clf.fit(X_train, y_train)
```

Out[20]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.9, n_estimators=200, random_state=42)
```

In [21]:
```
plot_decision_boundary(ada_clf, X, y)
```



In [22]:
```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1, random_state=42)

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

In [23]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

In [24]:
```
print(accuracy_score(y_test, y_pred))
```

```
0.92
```

In [25]:
```
print(accuracy_score(y_test, y_pred_rf))
```

```
0.912
```

In [26]:
```
np.sum(y_pred == y_pred_rf) / len(y_pred)
```

Out[26]:
```
0.976
```

In [27]:
```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1, 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 [28]:
```
rnd_clf.feature_importances_
```

Out[28]:
```
array([0.11249225, 0.02311929, 0.44103046, 0.423358  ])
```