Ensemble methods

Ensemble methods combine several base classifiers in order to improve their robustness when compared to their predictions alone. Several methods have been proposed in the machine learning literature. Scikit-learn provides us several classes to fit ensemble method, for example, VotingClassifier, BaggingClassifier, AdaBoostClassifier and RandomForestClassifier, to name a few. These classes will be explained in the sequence.

First we do all necessary imports, load the breast cancer dataset and define a method to plot a classifier's decision boundary.

In [1]:

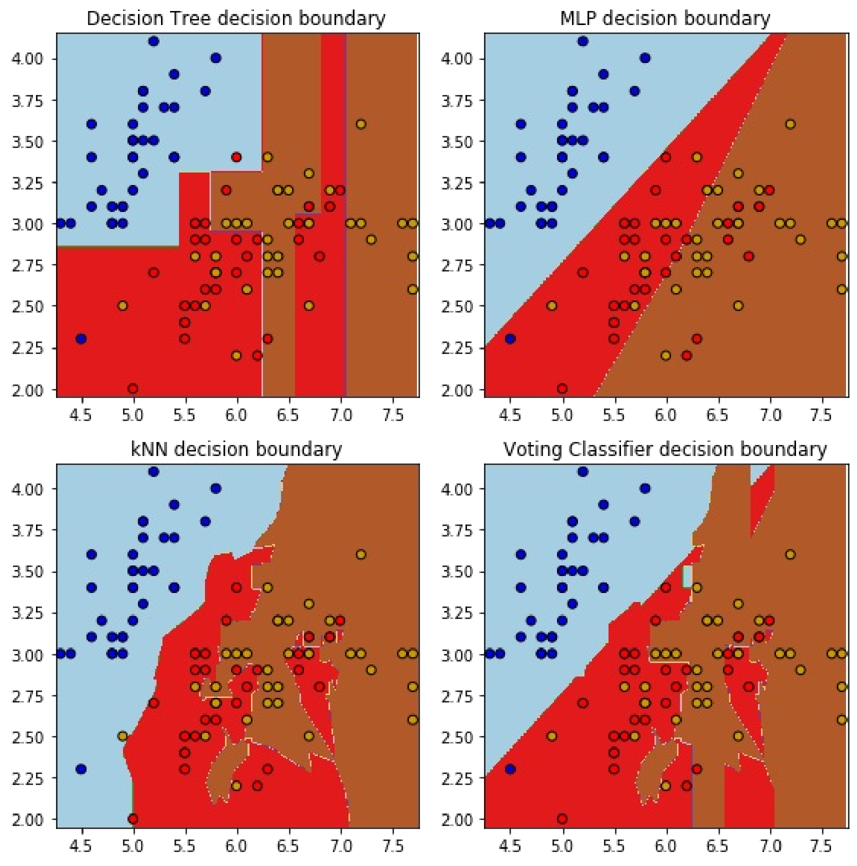
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import VotingClassifier, BaggingClassifier, AdaBoostClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Random seed.
seed = 10

# Loading Iris dataset.
data = pd.read_csv('data/iris.csv')

# Creating a LabelEncoder and fitting it to the dataset labels.
le = LabelEncoder()
le.fit(data['Name'].values)

# Converting dataset str labels to int labels.
y = le.transform(data['Name'].values)

# Extracting the instances data. In this example we will consider only the first two features to be able to
# plot the data and the decision boundaries of the classifiers.
X = data.drop('Name', axis=1).values[:, :2]

# Splitting into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.34, stratify=y, random_state=seed)

# Method to plot a classifier's decision boundary.
# This code is based on:
# http://scikit-learn.org/stable/auto_examples/semi_supervised/plot_label_propagation_versus_svm_iris.html
def plot_decision_boundary(classifier, X, y, title):
    xmin, xmax = np.min(X[:, 0]) - 0.05, np.max(X[:, 0]) + 0.05
    ymin, ymax = np.min(X[:, 1]) - 0.05, np.max(X[:, 1]) + 0.05
    step = 0.01

    xx, yy = np.meshgrid(np.arange(xmin, xmax, step), np.arange(ymin, ymax, step))

    Z = classifier.predict(np.hstack((xx.ravel()[:, np.newaxis], yy.ravel()[:, np.newaxis])))
    Z = Z.reshape(xx.shape)

    colormap = plt.cm.Paired
    plt.contourf(xx, yy, Z, cmap=colormap)

    color_map_samples = {0: (0, 0, .9), 1: (1, 0, 0), 2: (.8, .6, 0)}
    colors = [color_map_samples[c] for c in y]
    plt.scatter(X[:, 0], X[:, 1], c=colors, edgecolors='black')

    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)

    plt.title(title)
```

Voting classifier

The idea of the Voting Classifier is to combine different types of classifiers and to produce a prediction as the majority vote among them or the argmax of the mean probability of a class. In scikit-learn, this approach is implemented in the VotingClassifier class.

In [2]:

```python
plt.figure(figsize=(8, 8))

# Fitting a Decision Tree.
tree = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=3, random_state=seed)
tree.fit(X_train, y_train)
plt.subplot(2, 2, 1)
plot_decision_boundary(tree, X_train, y_train, 'Decision Tree decision boundary')

# Fitting a MLP.
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=10000, random_state=seed)
mlp.fit(X_train, y_train)
plt.subplot(2, 2, 2)
plot_decision_boundary(mlp, X_train, y_train, 'MLP decision boundary')

# Fitting a kNN.
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
plt.subplot(2, 2, 3)
plot_decision_boundary(knn, X_train, y_train, 'kNN decision boundary')

# Fitting a Voting Classifier by combining the three above classifiers.
voting_clf = VotingClassifier(estimators=[('Tree', tree), ('MLP', mlp), ('kNN', knn)], voting='hard')
voting_clf.fit(X_train, y_train)
plt.subplot(2, 2, 4)
plot_decision_boundary(voting_clf, X_train, y_train, 'Voting Classifier decision boundary')

plt.tight_layout()

plt.show()
```



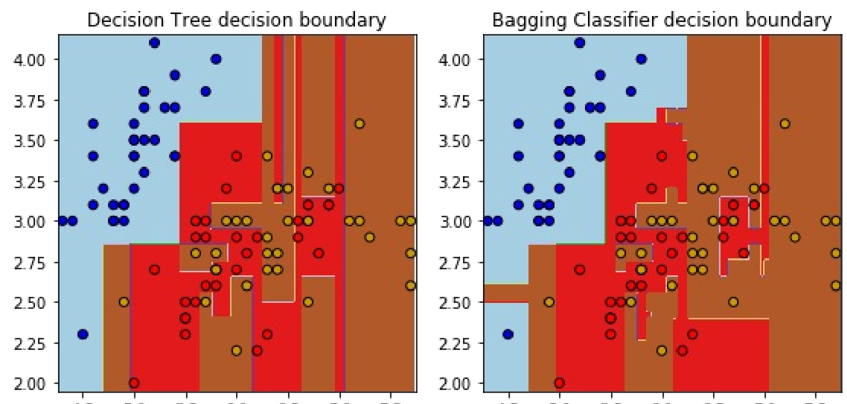
Bagging classifier

Bagging applies the same classifier on subsamples (usually with the same size) of the original dataset with replacement. In scikit-learn, this method is implemented through BaggingClassifier class. Its predictions return the label with highest mean probability among the base classifiers. If the base classifiers do not implement the predict_proba method, this class predicts the label by majority voting.

As mentioned in scikit-learn's documentation, the bagging method usually works well with more complex models (such as fully fitted decision trees).

In [3]:

```
plt.figure(figsize=(8, 4))

tree = DecisionTreeClassifier(random_state=seed)
tree.fit(X_train, y_train)
plt.subplot(1, 2, 1)
plot_decision_boundary(tree, X_train, y_train, 'Decision Tree decision boundary')

bagging_clf = BaggingClassifier(base_estimator=tree, n_estimators=50, random_state=seed)
bagging_clf.fit(X_train, y_train)
plt.subplot(1, 2, 2)
plot_decision_boundary(bagging_clf, X_train, y_train, 'Bagging Classifier decision boundary')

plt.tight_layout()

plt.show()
```



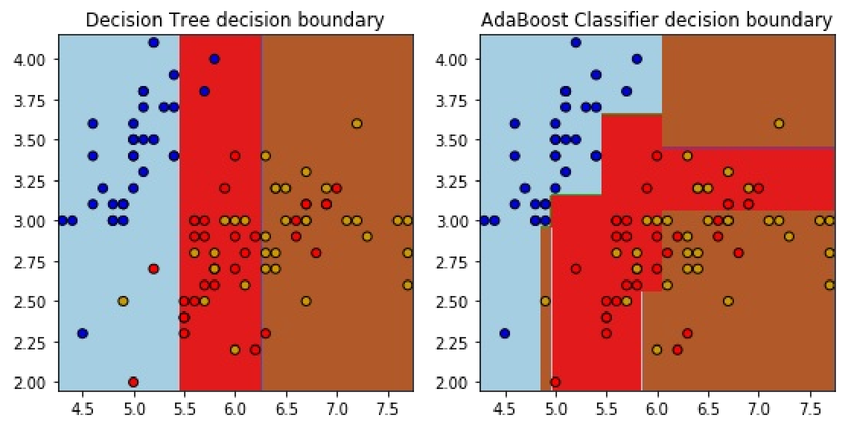
Boosting classifier

The Boosting method tries to combine several weak classifiers (i.e., classifiers that are slightly better than random classifiers) into a strong classifier. At each step, the procedure fits a new classifier with different weights on the objects from the training set. The idea is simple, objects that are assigned the wrong label will have their weights increased in the next iteration, while the others will have their weights decreased in the next iteration.

The most popular boosting algorithm of is AdaBoost. In scikit-learn, it is implemented in the AdaBoostClassifier class.

In [4]:

```
plt.figure(figsize=(8, 4))

tree = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=5, max_depth=3, random_state=seed)
tree.fit(X_train, y_train)
plt.subplot(1, 2, 1)
plot_decision_boundary(tree, X_train, y_train, 'Decision Tree decision boundary')

boosting_clf = AdaBoostClassifier(n_estimators=50)
boosting_clf.fit(X_train, y_train)
plt.subplot(1, 2, 2)
plot_decision_boundary(boosting_clf, X_train, y_train, 'AdaBoost Classifier decision boundary')

plt.tight_layout()

plt.show()
```



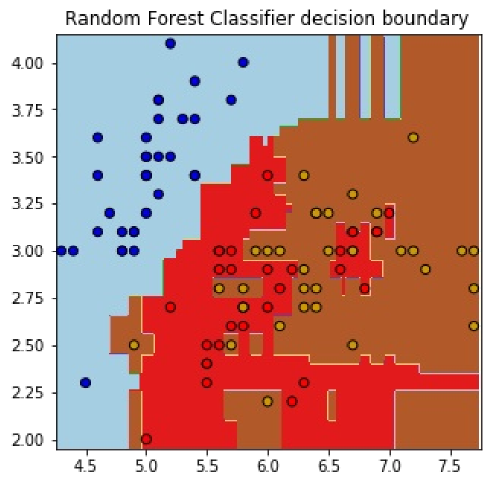
Random Forest classifier

Random Forest consists of an ensemble method composed by multiple decision trees. Each tree is trained with a subsample with replacement from the original dataset and, at each step, a node split is performed by choosing the best split among a random subset of the features instead of the best split overall.

Many experimental machine learning studies suggest that Random Forest is one of the best classifiers from the literature. In scikit-learn, this algorithm is implemented through RandomForestClassifier class.

In [5]:

```
random_forest_clf = RandomForestClassifier(n_estimators=50, random_state=seed)
random_forest_clf.fit(X_train, y_train)

plt.figure(figsize=(5, 5))
plot_decision_boundary(random_forest_clf, X_train, y_train, 'Random Forest Classifier decision boundary')
plt.show()
```


Random Forest Classifier decision boundary